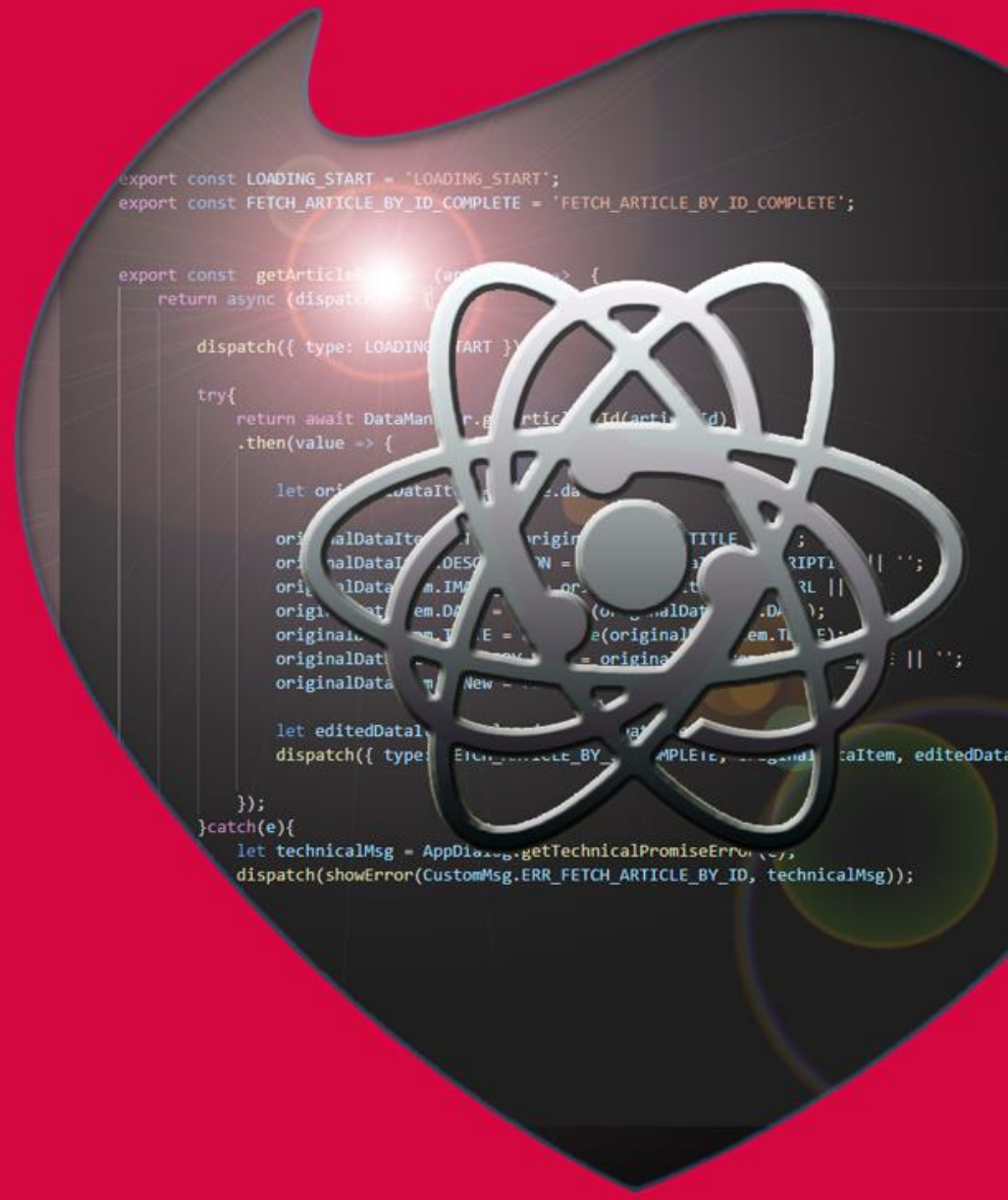
Domusvi

En toute confiance

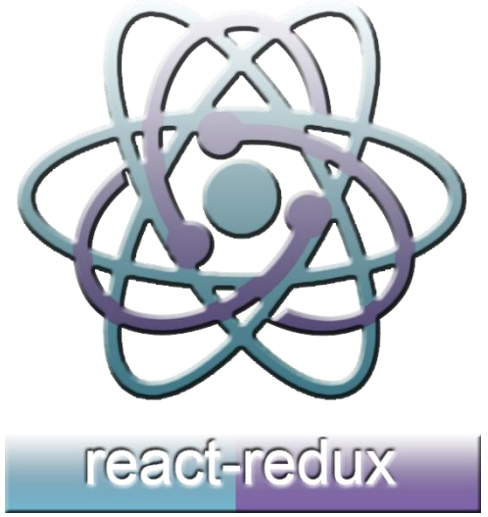
Formation React Avancée



Laurent TRAN BA

react-redux

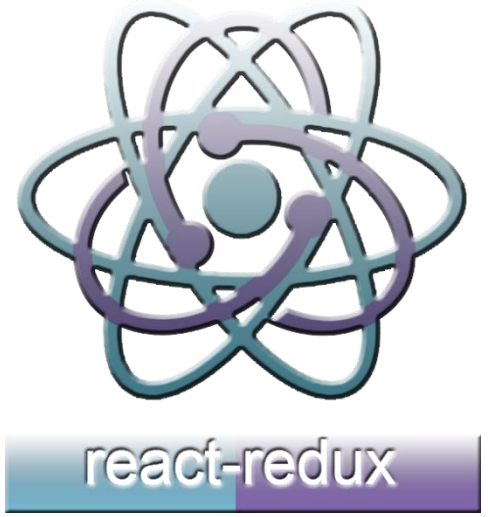
Sommaire (1/3)



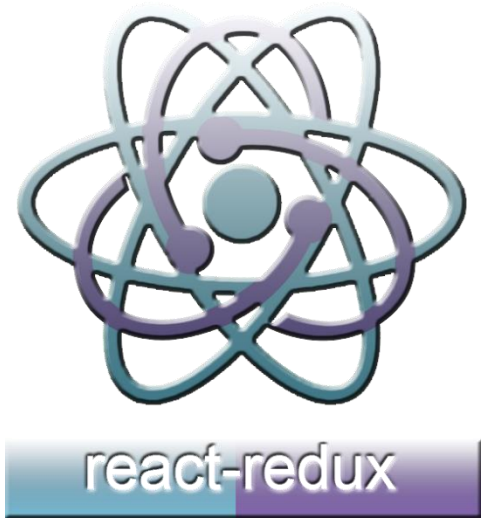
- **Rappels sur vos acquis**
  - Javascript / ES6
    - Fonctions fléchées (Arrow functions)
    - Manipulation des tableaux (map , find, forEach)
    - Propriété des Objects
    - Modules (import export)
    - Affectation par décomposition (destructuring)
    - Promises
    - Async await
    - Classes ,
  - React
    - Virtual DOM
    - React Component : state , render, props , stateful , stateless
    - Managemet du render JSX
    - React Component : cycle de vie et évènements
    - Diffusion du render entre les composants
    - Unidirectional Data Flow et Concept Lift State Up
    - React router
  - Architecture DomusVi
    - Front MVC ,emplacement du code React , référencement statique du bundle
    - React Router vs Routing MVC , pages de redirection MVC vs pages de redirection react
    - Npm : package.json
    - Configuration Webpack

Sommaire (2/3)

- **React-Redux**
  - Redux
    - Outils : installation de vscode et extensions, chrome et extensions
    - Redux – généralités
    - Redux adoption
    - Concepts de base
      - Store
      - Actions creator
      - Reducers
      - Architectures : rails,domain,ducks
    - Notion de middleware : thunk
    - Présentation du repository HitechStoreDemo
    - Exercice : compléter l'application
    - Review: Passage d'une application react sous redux : exemple store démo
    - Pour et contre : you might not need redux



Sommaire (3/3)



- **React-redux Mastering (advanced skills stage 1)**
  - Review sur redux-actions
  - Prop-type de default props (code)
  - Flow : typage javascript (code)
  - Tests Unitaires : Jest avec enzyme (code)
  - Tests Fonctionnels : Jest avec pupeeteer (code)
  - Nouveautés react 16,8 : hooks (code) , partage de state
  - Optimisation de performance : reduction des modification du DOM (shoulcomponentupdate , memoization)
  - Pattern avancés :
    - HOC,
    - portal,
    - context,
    - fragments,
    - render props
  - Design effects : csstransion , animate.css, pose (pop motion)
  - Immutabilité : Bad pratices
  - Internationalisation (voir code) : react-intl
  - ~~Custom Midleware~~
  - ~~Redux form~~

Rappel Javascript / ES6

Rappel

Liens vers la formation ½ :
https://domusvi.visualstudio.com/Portail%20Dvi/_wiki/wikis/Portail%20Dvi.wiki/146/-React-JS-Formation-(-2018-)

Ouvrir un compte sur codepen.io : pour effectuer les exercices

Les fonctions fléchées

Nous utiliserons les fonctions fléchées durant cette formation.
Ecrire la fonction multiply sous sa forme fléchée :

```
// écrire la fonction sous sa forme fléchée

let multiply = function (a,b){
    return a * b;
}
```

```
let multiply = (a, b) => { return a * b };
```

Rappel Javascript / ES6

Manipulation de tableaux d'objets : forEach,map

```
// 1 : utiliser forEach pour que les mots soient a1 b2 c3 d4
// 2 : simplifier en utilisant map pour stocker le résultat dans un tableau phrase

let mots = [
    { mot: "a" },
    { mot: "b" },
    { mot: "c" },
    { mot: "d" }
  ];
```

Réponses :

```
mots.forEach((m,idx) => (mots[idx].mot += idx));

console.log(mots);
```

```
let phrase = mots.map( (m,idx) => (m.mot += idx ) );

console.log(phrase);
```

Rappel Javascript / ES6

- **Manipulation de tableaux d'objets : every**

```
// utiliser every pour vérifier qu'aucun mot n'est vide

let mots = [
    { mot: "un" },
    { mot: "" },
    { mot: "trois" },
    { mot: "quatre" }
  ];
```

Réponse :

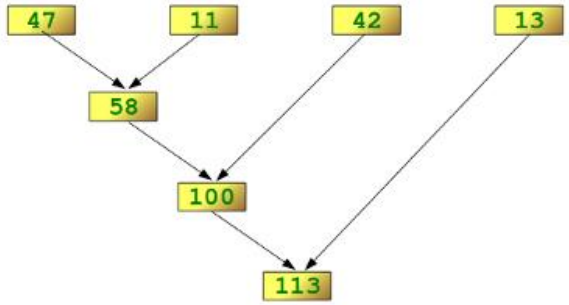
```
let isValid = mots.every((m) => (m.mot));

console.log(isValid);
```

Rappel Javascript / ES6

Manipulation de tableaux d'objets : reduce

```
// utiliser la fonction reduce pour obtenir 1 objet avec le mot : "utiliser la fonction reduce"
let mots = [
    { mot: "utiliser" },
    { mot: "la" },
    { mot: "fonction" },
    { mot: "reduce" }
  ];
```



Réponse :

```
let mot= mots.reduce((m1, m2) => ({ mot: m1.mot + " " + m2.mot}));

console.log(mot);
```

Rappel Javascript / ES6

rappels : les fonctions pour les tableaux :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

exercez vous chez vous:

https://www.w3resource.com/javascript-exercises/javascript-array-exercises.php

Rappel Javascript / ES6

Objets et propriétés

```
// Objet.keys renvoie un tableau des propriétés d'un objet
let properties  = Object.keys(myObject);
```

```
// Objet.entries renvoie un dictionnaire des propriétés/valeurs d'un objet
let propertieValues  = Object.entries(myObject);
```

```
// accès aux valeurs , nom de propriété variable

let prop = 'mot';
 myObject[prop] = 'e5';

 myObject.mot = 'e5';
```

Rappel Javascript / ES6

Modules ES6 : import export

Le système de module permet de gérer plus facilement les dépendances entre les différents modules de notre application. Au lieu de lister à plat l'ensemble des fichiers requis par notre application (dans l'ordre adéquat!) c'est le fichier Js principal qui indique les fichiers dont il dépend, qui eux même indiquent les fichiers dont ils dépendent et ainsi de suite. Ce mécanisme est particulièrement utile lorsqu'on développe des applications utilisant de nombreux fichiers et notamment des librairies tierces.

Noter que le chemin par rapport au root ~ est possible grâce au module: babel-plugin-root-import
et on peut simplifier en redéfinissant le root. voir code repo

1) Cas de l'import par défaut (un seul module exporté , un seul importé)

```
// fichier main.js

import myClass from "~/js/store/helpers/myClass.js";

// fichier myClass.js
export default class myClass  {
  // …
}
```

Rappel Javascript / ES6

Modules ES6 : import export

2) Cas de plusieurs modules exportés / importés : les exports nommés

```
// fichier clientHelper.jsx
export const customGet = (url) => { /*...*/ }
export const customPost = (url,data) =>{/*...*/}
export const customDelete = (url,id) =>{/*...*/}

// fichier dataSource.jsx
import {customGet,customPost,
customDelete } from  "~/js/store/helpers/clientHelpers.jsx";
```

Noter qu'il n'est pas possible d'écrire `export const default`...

Rappel Javascript / ES6

- **Affectation par décomposition (destructuring)**

```
const Collaborator = {
    firstName: Laurent,
    lastName: TRAN BA,
    job: Technical Director,
    id: 50254
};
const { firstName, lastName, job, id } = Collaborator;

console.log(firstName);
console.log(lastName);
console.log(job);
console.log(id);
```

il est possible d'extraire les variables d'un objet pour les lire directement

Rappel Javascript / ES6

- Promise : exercice chainage avec then

https://codepen.io/batran/pen/NWWbzLX?editors=1111

```
// soit une fonction qui retourne une promesse :

const getMessageAsync = (firstWord) => {
    return new Promise((resolve, reject) => {
        resolve(firstWord);
    });
  }

  // écrire le code qui : appelle la cette fonction et affiche :
  // "je"
  // "je suis"
  // "je suis pret"
  //  je , étant la valeur de retour de la promise dans l'expression "je suis",  "je suis" est le retour dans "je suis pret"
  //  indice : utiliser le chainage
```

Rappel Javascript / ES6

- Promise : exercice - réponse

```
getMessageAsync('je').then((response)=>{
    console.log(response);
    return response + ' suis';

  }).then((response)=>{
    console.log(response);
    return response +' pret';

  }).then((response)=>{
    console.log(response);
  });
```

Rappel Javascript / ES6

Promise : méthodes statiques utiles et fonction thenable

Une fonction « thenable » est une fonction qui retourne le résultat d'une promise, par conséquent on peut l'utiliser avec l'opérateur then.

Il est donc parfois pratique de pouvoir retourner immédiatement le résultat d'un promise dans une fonction.

La méthode promise.resolve() permet cela :

```
const getMyStuffAsync = () =>{

    return Promise.resolve().then(()=>{
        return 'to be continued...'
    });
}

getMyStuffAsync()
.then((response) =>{
    console.log(response);
});
```

autres fonction statiques (Promise.all , Promise.race, Promise.reject)

Rappel Javascript / ES6

Promise : async et await , une nouvelle écriture

Une autre écriture permet simplifer la gestion de l'écriture asychrone .
Elle est ressemblante à celle en C#
soit un echainement d'appel asynchrone avec des promises : https://codepen.io/batran/pen/QWWZyPZ

```
//soit une methode qui renvoie une promesse
const getMessageAsync = (msg) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(msg + '...done !');
        }, 800);
    })
}
const main = () => {
    let message = 'Sending!';
    console.log(message);
    getMessageAsync(message)
        .then((msg) => {
            console.log(msg);
            getMessageAsync(msg)
                .then((msg2) => {
                    console.log(msg2);
                })
        });
}

main();
```

Rappel Javascript / ES6

- **Promise : async et await , une nouvelle écriture**

Avec async await cela s'ecrit plus simplement (séquentiellement) :

https://codepen.io/batran/pen/NWWONym

```
//soit une methode qui renvoie une promesse
const getMessageAsync = (msg) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(msg + '...done !');
        }, 800);
    })
}
const main = async () => {
    let message = 'Sending!';
    console.log(message);
    const response1 = await getMessageAsync(message);
    console.log(response1);
    const response2 = await getMessageAsync(response1);
    console.log(response2);
}

main();
```

Rappel Javascript / ES6

Classe ES6

```
class Rectangle {
    constructor(hauteur, largeur) {
      this.hauteur = hauteur;
      this.largeur = largeur;
    }
    zoom = (zoomfactor) =>{
       this.hauteur *= zoomfactor;
       this.largeur *= zoomfactor;
    }
  }

 const p = new Rectangle(); // ReferenceError
 const r = new Rectangle(20,40); // ok
```

NB : Grace aux fonctions fléchées , vous n'avez pas besoin d'effectuer :

this.zoom = this.zoom.bind(this) , dans le constructeur.

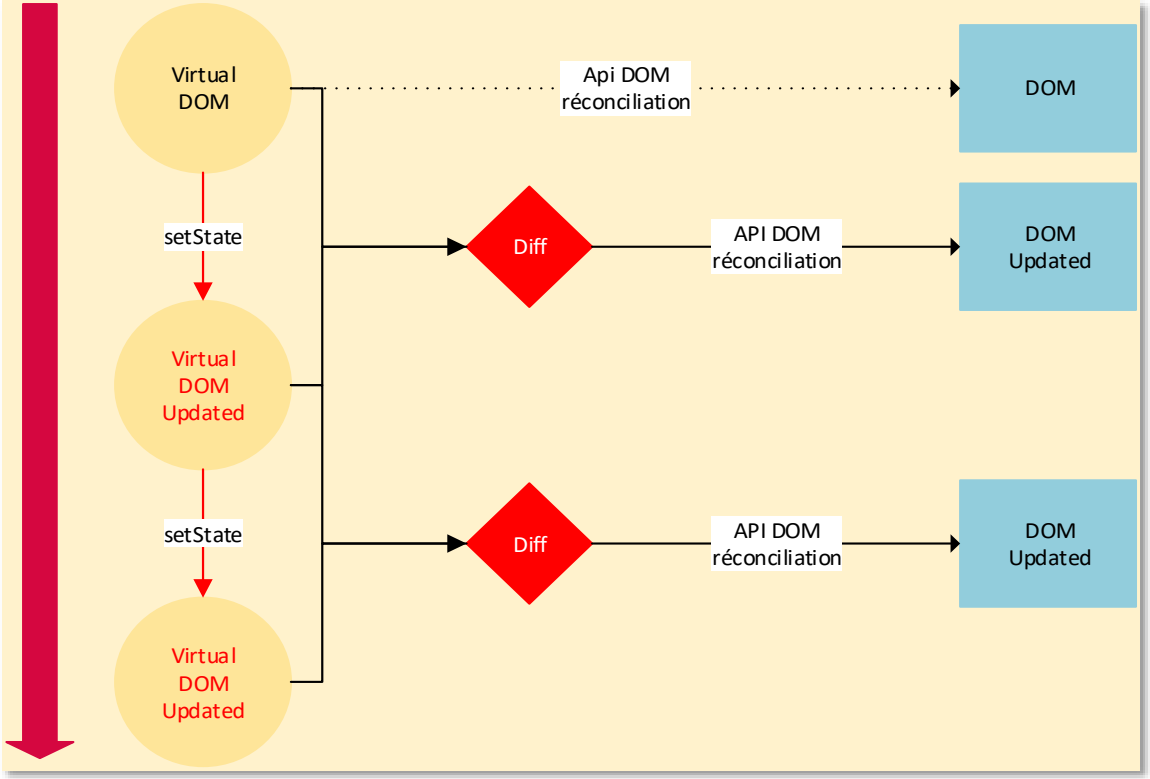
(par default le scope de this est la fonction locale elle-même)

Rappel React

Virtual DOM

Crée en 1998 , le DOM n'a pas été conçu pour encaisser autant de changement qu'une application d'aujourd'hui l'exige; l'api DOM est très couteuse en performance.

Le DOM virtuel (objet javascript) permet de récolter les demandes de changement et d'appliquer uniquement le différentiel entre les arbres.(réconciliation, parcours O(n) aidé par des heuristiques*).



Heuristiques : Pour que cela fonctionne de manière optimale :

* React considère que : Deux éléments de types différents produiront des arbres différents.

* Le développeur aide react en indiquant quels éléments peuvent être stables d'un rendu à l'autre grâce à la prop key. (ex: fonction map dans les render)

Rappel React

Classe React stateless

- Une classe stateless n'a pas de state, contrairement à un classe stateful, les paramètres sont passées par props.
- Par convention les évènements commencent par handle (plutôt que on : handleClick au lieu de onClick)

```
class myForm extends React.Component{

    constructor(props) {
        super(props);
    };

    handleClick = ()=>{
        console.log('click');
    }

    render(){
        return(
            <div>
                {this.props.myText}
                <button onClick={this.handleClick}>test</button>
            </div>
        );
    }
}
```

la classe myForm pourra être réutilisée dans le bloc render d'un composant parent: <myForm myText='Hello World'>

Rappel React

Classe React stateful

https://codepen.io/batran/pen/yLLVxjM

Le state est un Objet : {}, **immutable**.
Il n'est autorisé de mettre à jour le state que par la méthode setState(), qui procure la traçabilité des changement su state.

Chaque changement du state déclenche le réaffichage du composant (la méthode render est appelée).
Les enfants sont aussi signifiés que le parent est réaffiché.

La méthode render est appelée aussi après la construction du Composant.

Dans le constructeur le state est initialisé Par affectation directe.

Dans les autres méthodes, la méthode **setState** permet de déclencher le réaffichage du composant (render())

```
class MyForm extends React.Component{

        constructor(props) {
            super(props);
            // seul endroit ou l'affection directe du state est permis
            this.state={myText :'Hello World !'}
        };

          handleClick = ()=>{
                    // modification du state => render() :
                    // (attention setState est asynchrone)
                    this.setState({
                        myText:'Thanks for your like !'
                    },
                     ()=> {console.log(this.state)}
                    );
                }

          render(){
                    return(
                        <div>
                            {this.state.myText}
                            <button onClick={this.handleClick}>like</button>
                        </div>
                    );
                }
}


ReactDOM.render( <MyForm />, document.querySelector( '.container' ) );
```

Rappel React

Management du render JSX

JSX pour ***JavaScript XML*** est une forme de syntaxe pseudo XML.

- La méthode render renvoie un et seul nœud JSX de départ. (vous obtiendrez une erreur si vous en retourner plus d'un)

```
render(){
        return(
            <div>
                <div>Titre</div>
                <div>Sous titre</div>
            </div>
        );
    }
```

- Pour faire appel à des variables dans le bloc de retour, utiliser les accolades {}

```
render(){
        return(
            <div>
                <div>{this.props.title}</div>
                <div>{this.props.subtitle}</div>
            </div>
        );
    }
```

Rappel React

Management du render JSX : boucles

En Jsx , il n'existe pas de système de boucle « for ou while »

Pour boucler utiliser la méthode map directe, ou passer par une fonction temporaire.

exercice compléter la partie jsx manquante : https://codepen.io/batran/pen/GRRNwry

```
class MyForm extends React.Component{

        constructor(props) {
            super(props);
            // seul endroit ou l'affection directe du state est permis
            this.state={mots : [
                        { mot: "un" },
                        { mot: "deux" },
                        { mot: "trois" },
                        { mot: "quatre" }
                      ]}
        };

   render(){
            return(
                <div>
                  {/*compléter*/}
                </div>
            );
        }
}

ReactDOM.render( <MyForm />, document.querySelector( '.container' ) );
```

Rappel React

Management du render JSX : boucles

- Réponses

```
render(){
        return(
            <div>
                {this.state.mots.map( (m,idx) => {
                  return <div key={idx}>{m.mot}</div>
                }) }
            </div>
        );
    }
```

```
render(){
 const mots = this.state.mots.map( (m,idx) => {return <div key={idx}>{m.mot}</div>});
    return(
        <div>
            {mots}
        </div>
    );
 }
```

Rappel React

Management du render JSX : conditions

- En Jsx , il n'existe pas de système de boucle « if ou else if »

Pour utiliser une condition utiliser soit : un opérateur ternaire , soit un opérateur boolean (&&), soit une fonction qui gèrera l'affichage.

Exercice facultatif : compléter la partie jsx manquante : https://codepen.io/batran/pen/zYYoMPr

Attention utiliser un affichage conditionnel au niveau jsx implique que vous détruisez le composant quand celui-ci n'est plus affiché.

Si vous ne désirez pas détruire le Composant , déporter la gestion de l'affichage dans le composant enfant ,

Avec par exemple une prop display..

Rappel React

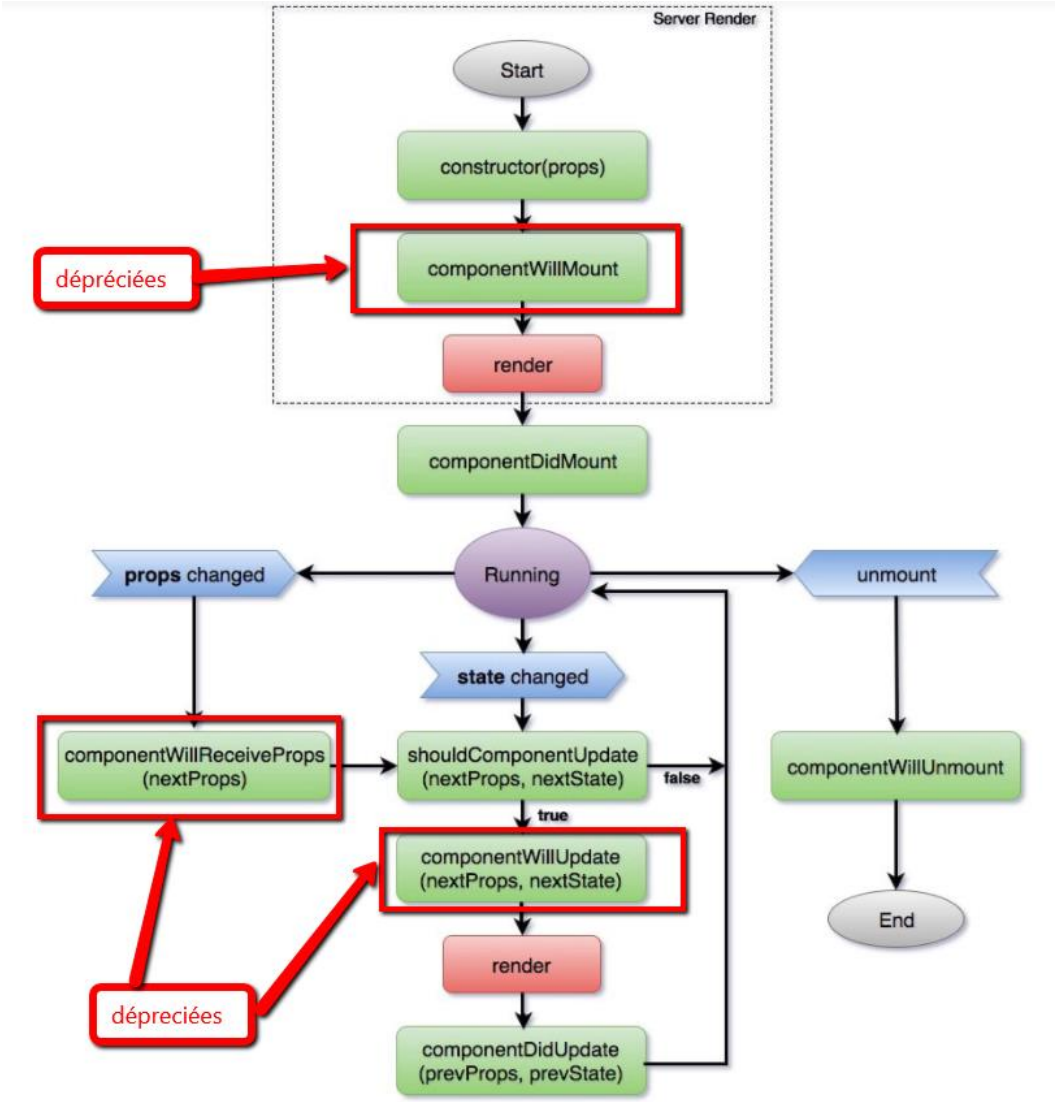
- **Cycle de vie**
  - **Ancien cycle de vie (deprecated)**

Le schéma de droite montre le cycle de vie
et les méthodes disponibles durant ce cycle
dans une classe React.

Les méthodes **componentWillMount**,
componentWillReceiveProps, **componentWillUpdate**
sont maintenant dépreciée dans les versions actuelles
de React car elles ne sont pas theadSafe

Elles auront disparue en version 17, nous sommes
à la version 16.10
Et seront maintenue avec le prefixe UNSAFE_ :

- **UNSAFE_componentWillMount**
- **UNSAFE_componentWillReceiveProps**
- **UNSAFE_componentWillUpdate**



Rappel React

- Cycle de vie
  - Nouveaux cycle de vie

http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

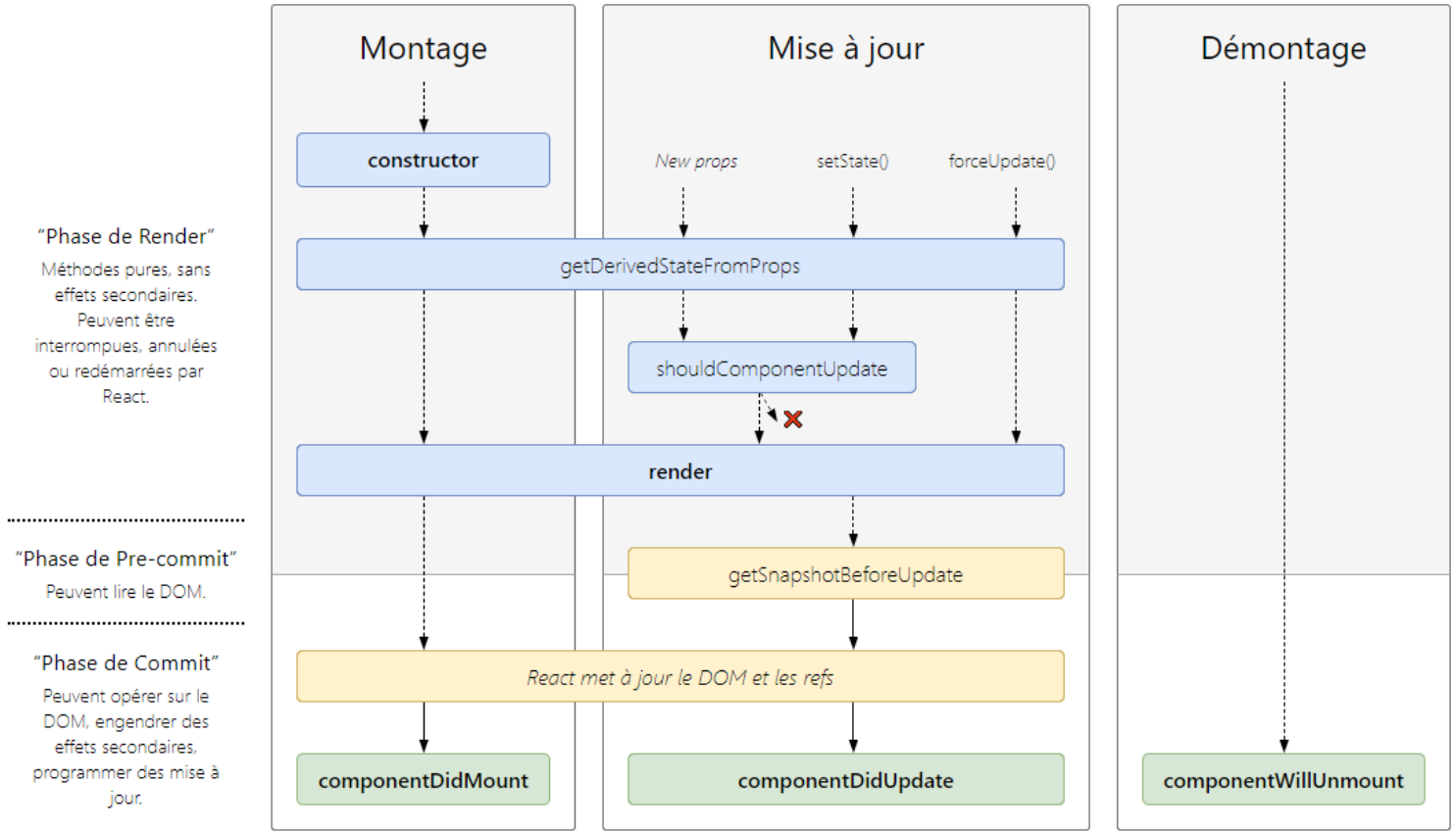
Dans le nouveau cycle,
On trouve 2 nouvelles méthodes :

static getDerivedStateFromProps

(la méthode étant static vous ne pourrez pas acceder au this de la classe)
Utilisation rare, voire plutôt componentDidUpdate

getSnapshotBeforeUpdate

Pour remplacer componentWillUpdate



Rappel React

Diffusion de l'évènement Render dans l'arbre des composants

Lorsqu'un composant a sa méthode render() appelée, il se redessine et diffuse à ses composants enfants l'information de se redessiner.

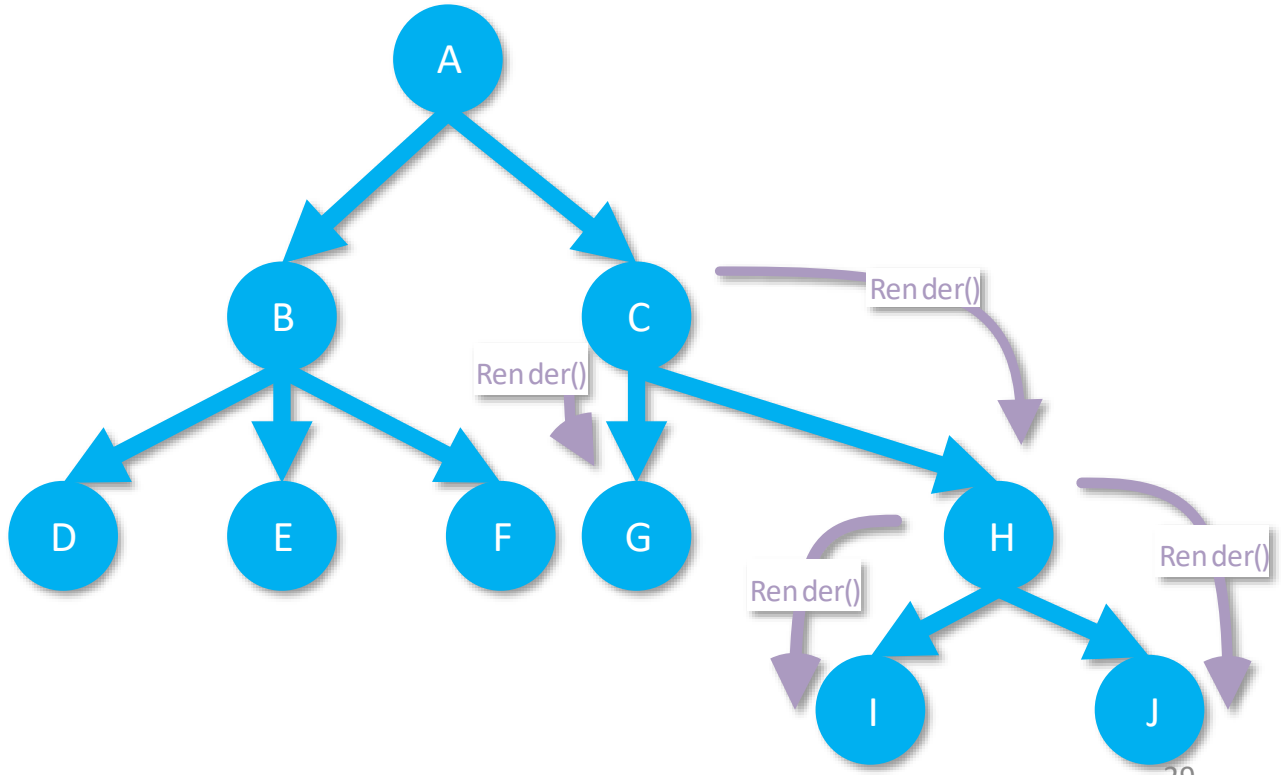
Chaque enfant ayant reçu le signal du parent se redessinera à son tour et transmettra le signal à ses propres enfants, si et seulement si la méthode **shouldComponentUdpate** retourne true. (par default, elle retourne true)

On Comprend que plus les composants sont stateful, moins ce mécanisme est optimal.
De plus l'état global des states devient imprévisible (unpredictible state)

C'est pourquoi React nous conseille de remonter les propriété du state du composants dans le parent.
C'est le concept « lift state up »

Est ce que la conception stateful est une bad pratice Oui ? Non ? Dans certains cas ?

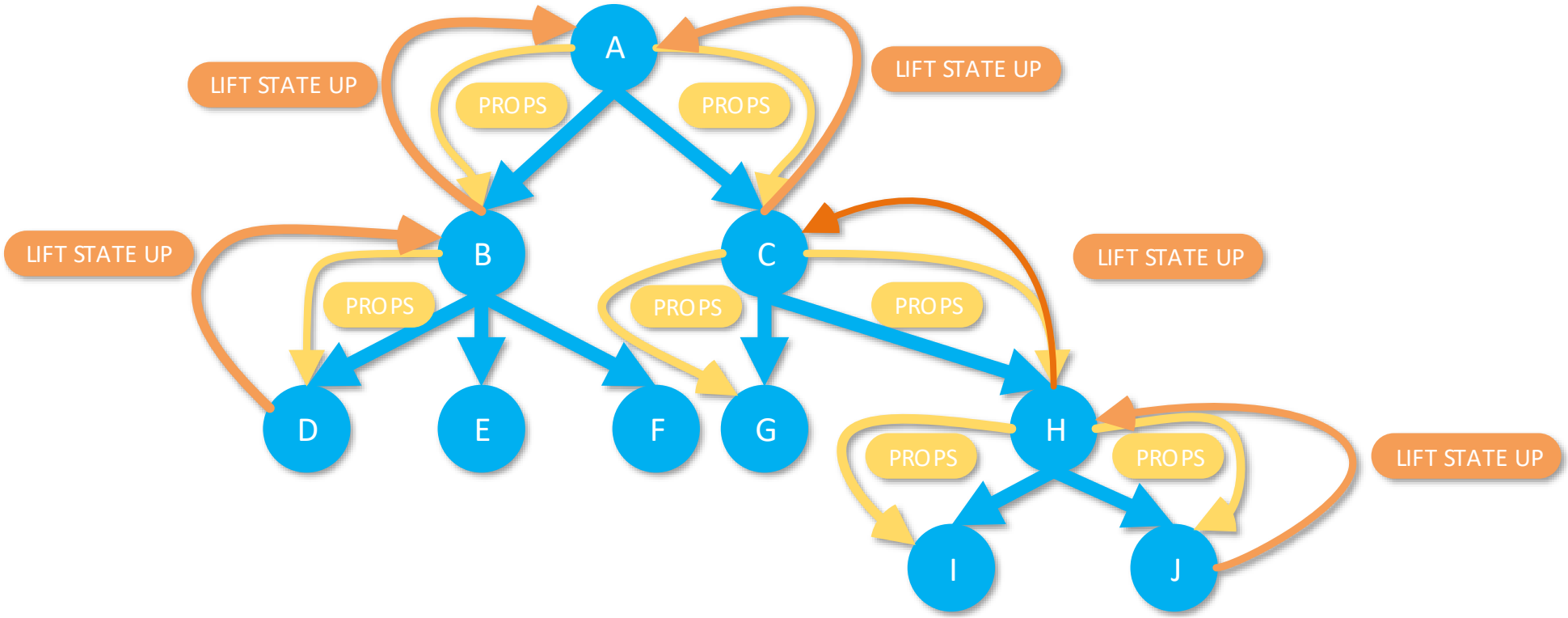
« Lift State Up » a-t-il des inconvénients ?



Rappel React

Unidirectional Data Flow et « Lift State Up »

Le principe de react est « Unidirectional Data Flow » , les données circulent dans un seul sens.
Le parent transmet les données nécessaires au Composant fil via les props.
+ Si l'on suit le principe « Lift State Up » le parent gère toutes props de tous ces enfants et sous-enfants :



Rappel React

Inconvévient du « Lift State Up »

=> Si l'on suit le principe « Lift State Up » le parent gère toutes props de tous ces enfants et sous-enfants :
Cela signifie aussi qu'il faut transporter et distribuer les props (attributs+ évenements) de tous les enfants par blocs sur tout l'arbre. Cela peut au final devenir très vite illisible, voire ingérable : (extrait du repo residence) …

```
<ContentDetail idNavActive={this.state.idNavActive}

                generalViewModel={this.generalViewModel}
                generalViewModelStorage={this.generalViewMode.Stor e}
                resetGeneral={() => { return this.initGeneralVi  Mo l(this.state.da  Es tablishm  t)
                validateGeneral={this.validateEstablishmentViewM del}
                dataTypesPerimeters={this.state.dataTypesPerimete }
                dataSegments = {this.state.dataSegments}

                legalViewModel={this.legalViewModel}
                infosCompanyLegalViewModel={this.state.dataCompanyLeg }
                resetLegal={() => { return this.initLegalViewModel(this.stat  dataEst  ment)}}
                validateLegal={this.validateEstablishmentViewModel}
                dataLegalCcompanies={this.state.dataLegalCompanies}
                legalCompany={this.state.legalCompany}

                internetViewModel={this.internetViewModel}
                resetInternet={ () => { return this.initInternetViewModel(th .s  .stabl
                validateInternet={this.validateEstablishmentViewModel}
                dataGeoPerimeterGroup={this.state.dataGeoPerimeterGroup}
                dataTypeGeoPerimeterGroup={this.state.dataTypeGeoPerime erGroup}
                updateFrontPhoto={this.updateFrontPhoto}
                //// coupé au 1/3 pour affichage ……..>>>>>>>>>>
/>
```

Redux pourra peut être nous apporter la solution …

Rappel React

- **React-router :** https://reacttraining.com/react-router/web/guides/quick-start

- react-router vous permet de charger un composant React en fonction de l'url et ceci sans déclencher d'appel serveur.
- react-router offre la possibilité de gérer les paramètres dans la route et même depuis récemment de les typer à l'aide des expressions régulières. Les composant routé doivent être décoré à l'aide du composant withRouter.

```
import { BrowserRouter as Router,Switch,Route,withRouter} from 'react-router-dom';
ReactDOM.render(
        <Router basename = '/myapp.webclient/formation'>
                    <Switch>
                        <Route exact  path="/formations" component={ListFormations} />
                        <Route path={"/formations/:id(\\d+)"} component={FormationEdit} />
                        <Route path={"/formations/new"} component={FormationEdit} />
                        <Route exact path="/401"   component={ForbiddenPage}  />
                        <Route component={NotFoundPage} />
                    </Switch>
         </Router>
);

// dans le fichier ListFormations.jsx
export default withRouter(ListFormations);

class listFormations extends React.Component{
    //...
    componentDidMount(){
        let formationId = this.props.match.params.id; // permet de récuperer le paramètre id
    }
}
```

Rappel React

React-router

- Lien hypertexte avec le Composant **Link** (et <a href>

```
return (
    <td>
        <Link to={url} >Formation React avancée</Link>
    </td>
);
```

- navigation dynamique avec **history.push()** et non windows.location

```
AppDialog.setConfirm(CustomMsg.INFO_PAGE_CHANGE, null, () => {
    this.resetData();
    document.title = "Gestionnaire du magasin d'articles";
    this.props.history.push('/articles');
});
```

Rappel Architecture DomusVi , configuration

- **Architecture mixte MVC + React/webpack**
  - Rôle architecture MVC très limité à la sécurité et l'obtention du token, ainsi qu'au déploiement.
  - La partie source React est dans le répertoire ….webclient\Scripts\src
  - La partie compilée/distribuable (par webpack) est dans le répertoire ….webclient\Scripts\dist
  - MVC référence la partie js compilée dans le fichier :

```
public static void RegisterBundles(BundleCollection bundles)
        {
            BundleTable.EnableOptimizations = false;

            #region WEBPACK

            bundles.Add(new ScriptBundle("~/bundles/app").Include(
                    "~/Scripts/dist/app.formation.bundle.js"
                ));
            #endregion
```

  - La gestion du token permettant d'interroger les api DomusVi est géré par la master au niveau du helper js Apihelper

```
class ApiHelper {
    //Permet de d'obtenir le token et configurer axios pour une requete get
    //url : url de la ressource
    static Get(url) {
        let auth_token = master.auth.getValidToken(); // obtention d'un token valide via la master page
        axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth_token;
        axios.defaults.timeout = timeout;
        return axios.get(url);
    }
```

Rappel Architecture DomusVi , configuration

React Router vs Routing MVC

- Le routing MVC se fait jusqu'à la partie de l'url correspondant au controller MVC

  ex: https://localhost/formation.webclient/formations où Formation est le controler FormationController

- Pour pouvoir laisser ensuite le contrôle à react-router il est nécessaire de configurer le fichier RouteConfig.cs

```
public class RouteConfig
   {
       public static void RegisterRoutes(RouteCollection routes)
       {
            //  …
           routes.MapRoute(
              name: « FormationsRoute",
              url: « formations/{*otherArgs}",
              defaults: new { controller = “Formations", action = "Index" });
```

Rappel Architecture DomusVi , configuration

Pages de redirection MVC

- Pour MVC : Configurer les pages de redirection 404,… via le fichier web.config

```
<system.web>
   <customErrors mode="On" >
      <error statusCode="404" redirect="~/Formations/Error/404" />
    </customErrors>
</system.web>
```

- Et ajouter les routes de redirections dans le BaseController : (ajouter bien sûr les views)

```
public class BaseController : Controller
   {
       [Route("Error/401")]
       public ActionResult Error401()
       {
           return View("~/Views/Shared/Forbidden401.cshtml");
       }
       [Route("Error/404")]
       public ActionResult Error404()
       {
           return View("~/Views/Shared/NotFound404.cshtml");
       }
   }
```

Rappel Architecture DomusVi , configuration

Pages de redirection React

- Pour React-router: Utiliser simplement les routes avec switch et,à l'instar des view en MVC, créer un composant react pour chaque redirection.

```
render(){
        return(
                <div>
                    <Switch>
                        <Route exact  path="/articles" component={ArticleList} />
                        <Route path={"/articles/:id(\\d+)"} component={ArticleEdit} />
                        <Route path={"/articles/new"} component={ArticleEdit} />
                        <Route exact path="/401"   component={ForbiddenPage}  />
                        <Route component={NotFoundPage} />
                    </Switch>
                </div>
        )
    }
```

Rappel Architecture DomusVi , configuration

npm : configuration package.json

- Npm est à javascript ce que Nuget est à C# , c'est la bibliothèque internationale des modules js.!
- La racine d'un projet js commence là où le fichier package.json est installé.
- Pour initialiser un nouveau projet et créer un fichier package.json , utiliser la commande:

```
npm init
```

- Le fichier package.json référence tous les packages npm que vous avez installé mais pas que.
- Il peut aussi contenir: vos commandes de compilation webpack
- Quand vous installez un package vous pouvez le déclarer soit :
  - En tant que dépendance de développement : les packages de developpement ne sont pas utiles au fonctionnement de l'application sur le serveur et s'installe comme suit

```
npm i mon-package --save-dev
```

  - En tant que dépendance : ils s'installe sans la commande --save-dev

```
npm i mon-package
```

pour installer une version spécifique :

```
npm i mon-package@12.3.1
```

Rappel Architecture DomusVi , configuration

Webpack : configuration (1/3)

- Vous devez d'abord installer webpack sur votre application

```
npm i webpack
```

- Le fichier package.json doit entre ensuite configuré :
  - Le fichier main de l'application doit être précisé :

    "main": "js/formations/index.jsx",

  - Les commandes de compilation doivent être ajoutées et pointer vers le ou les fichiers de config webpack:

```
"main": "js/formations/index.jsx
 "scripts": {
      "start": "webpack --watch --mode=development --config webpack.dev.js",
      "dev": "webpack --mode=development --config webpack.dev.js",
      "prod": "webpack --config webpack.prod.js",
      "test": "echo \"Error: no test specified\" && exit 1"
}
```

cela vous premettra de lancer les commande : **npm run start**, etc..

Rappel Architecture DomusVi , configuration

Webpack : configuration (2/3)

- Vous devez ensuite préparer vos fichiers de configuration webpack.*.js
- Par exemple, vous pouvez gérer 1 fichier commun à la dev et la prod , puis 2 fichiers complémentaires, spécifique à la dev et la production.
- **Webpack.common.js** : il contiendra , les entries, le nom du fichier compilé

```
const path = require('path');

module.exports = {
    // Fichier d'entrée
    entry: {
        formations :'./js/formations/index.jsx'
    },
    // Fichiers de sorties [name] correspond au paramètres entry ci-dessus
    output: {
        path: path.resolve(__dirname, '../dist/'),
        filename: 'app.[name].bundle.js'
    },
    module: {
        rules: [
            {
                test: /\.(js|jsx)$/, // tous les fichiers .js
                exclude: /node_modules/, // sauf le dossier node_modules
                use: { // seront transpilés par babel
                    loader: 'babel-loader',
                    options: {
                        cacheDirectory: true // accélère la génération des JS en cachant les transpilations
                    }
                }
            }
        ]
    }
}
```

Rappel Architecture DomusVi , configuration

Webpack : configuration (3/3)

- webpack.dev.js : il permettra de définir le mode dev de react, ainsi que le source-mapping

```
const merge = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
   mode: 'development',
   devtool: 'inline-source-map',
   devServer: {
     contentBase: '../dist'
   },
   devtool: 'source-map',
   externals : {
        config: "dev_config"
   }
});
```

- webpack.prod.js : il permet de définir le mode production de react plus performant

```
const merge = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
   mode: 'production',
   externals : {
        config: "prod_config"
   }
});
```

React-Redux

React State

Dans un arbre de composants,

le déclenchement des rendus

rend la prédiction des états de rendu complexe

En effet pour un composant

L'ordre de mise à jour peut venir

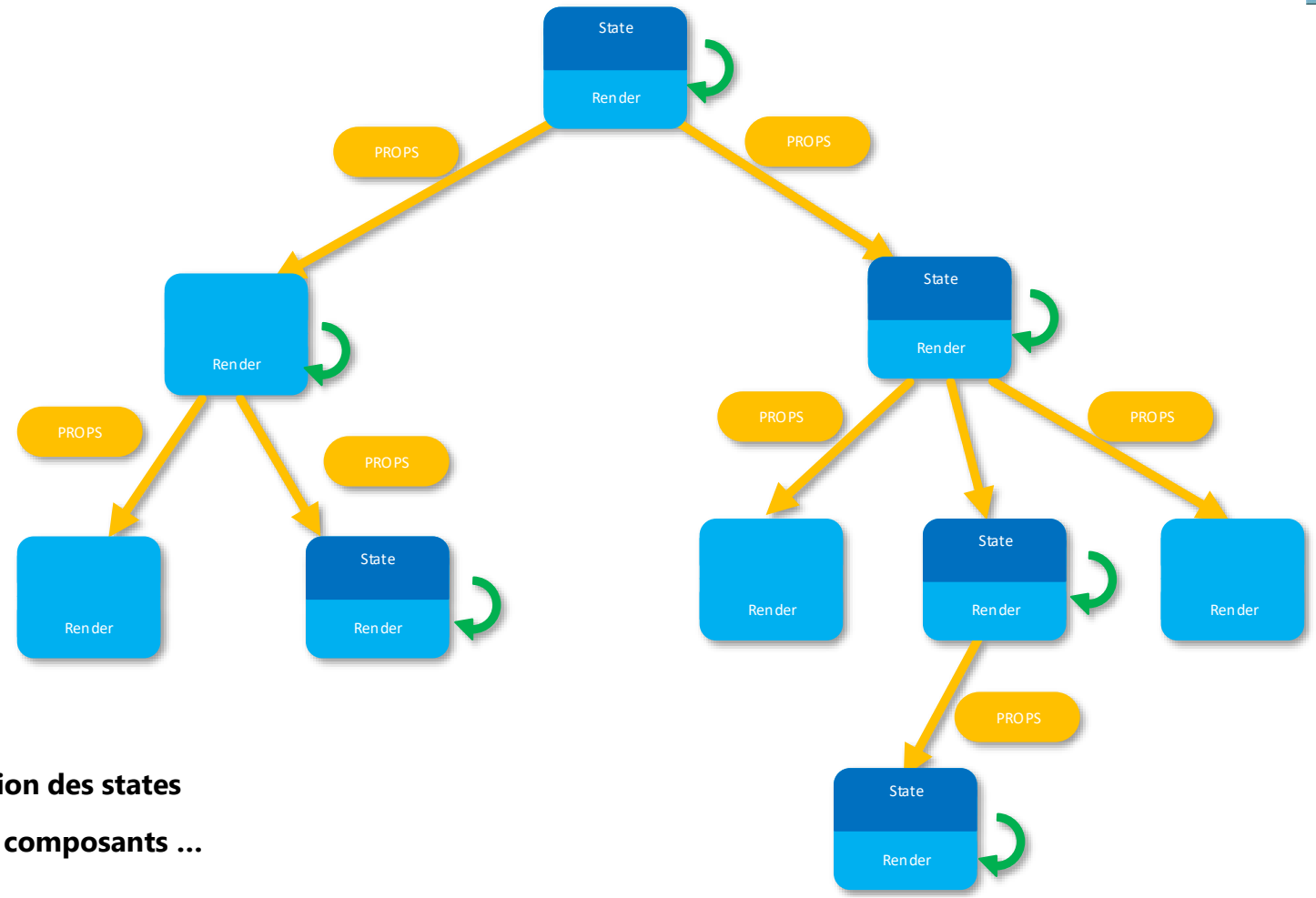
d'un changement de son state comme

de celui d'un de ses parents.

Rendre tous les composants « stateless »

rend parfois le code non maintenable

(volumétrie des paramètres)



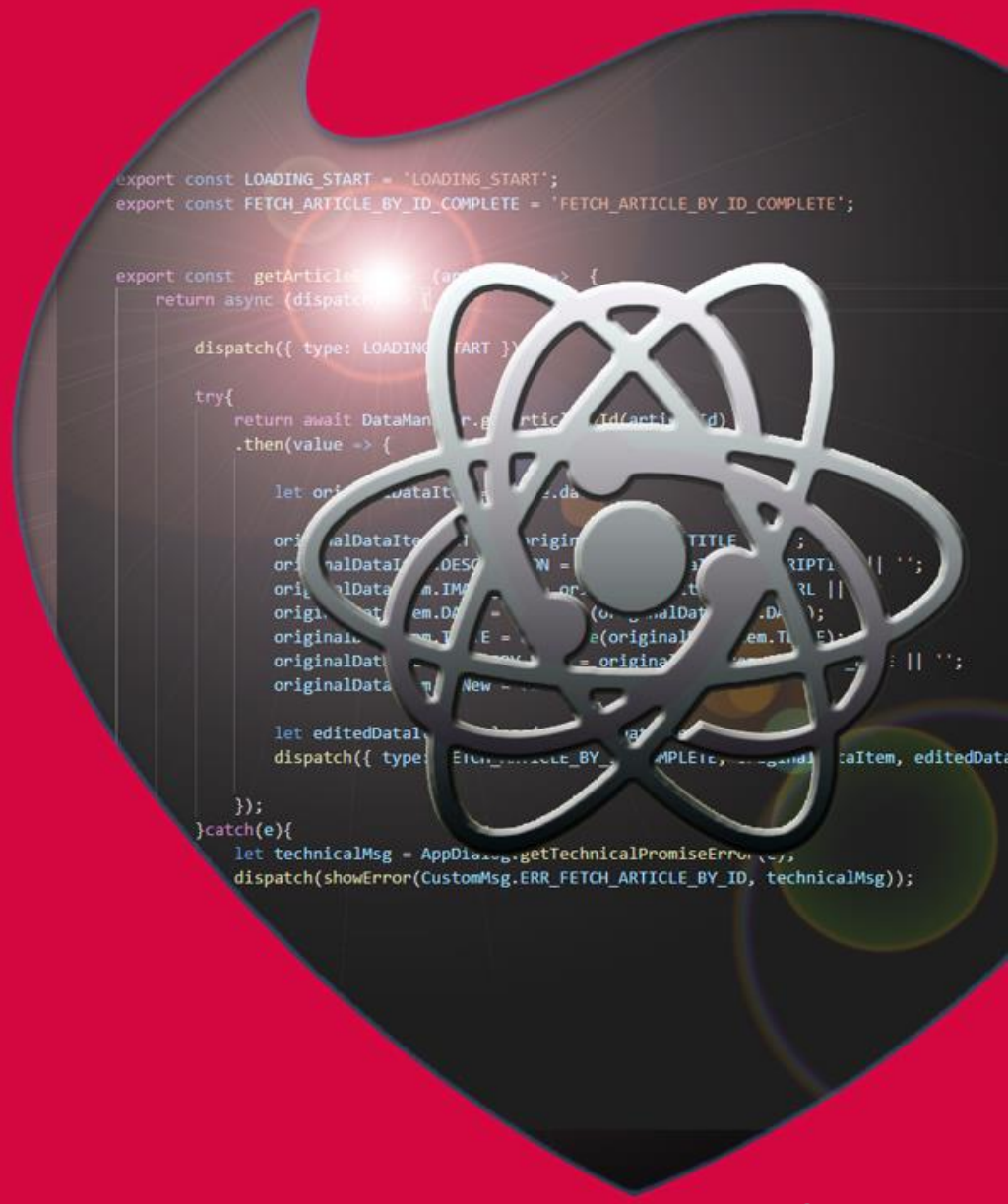
Une solution serait de pouvoir centraliser la gestion des states

Dans un même endroit et le rendre accessible au composants …

Domusvi

En toute confiance

Formation React Avancée Pause



Laurent TRAN BA

react-redux

Outils

Trucs utiles

Mettre à jour npm : npm install -g npm

Mettre jour tous les package npm d'un projet : npm update (et sa limite)

Améliorer son environnement vscode :

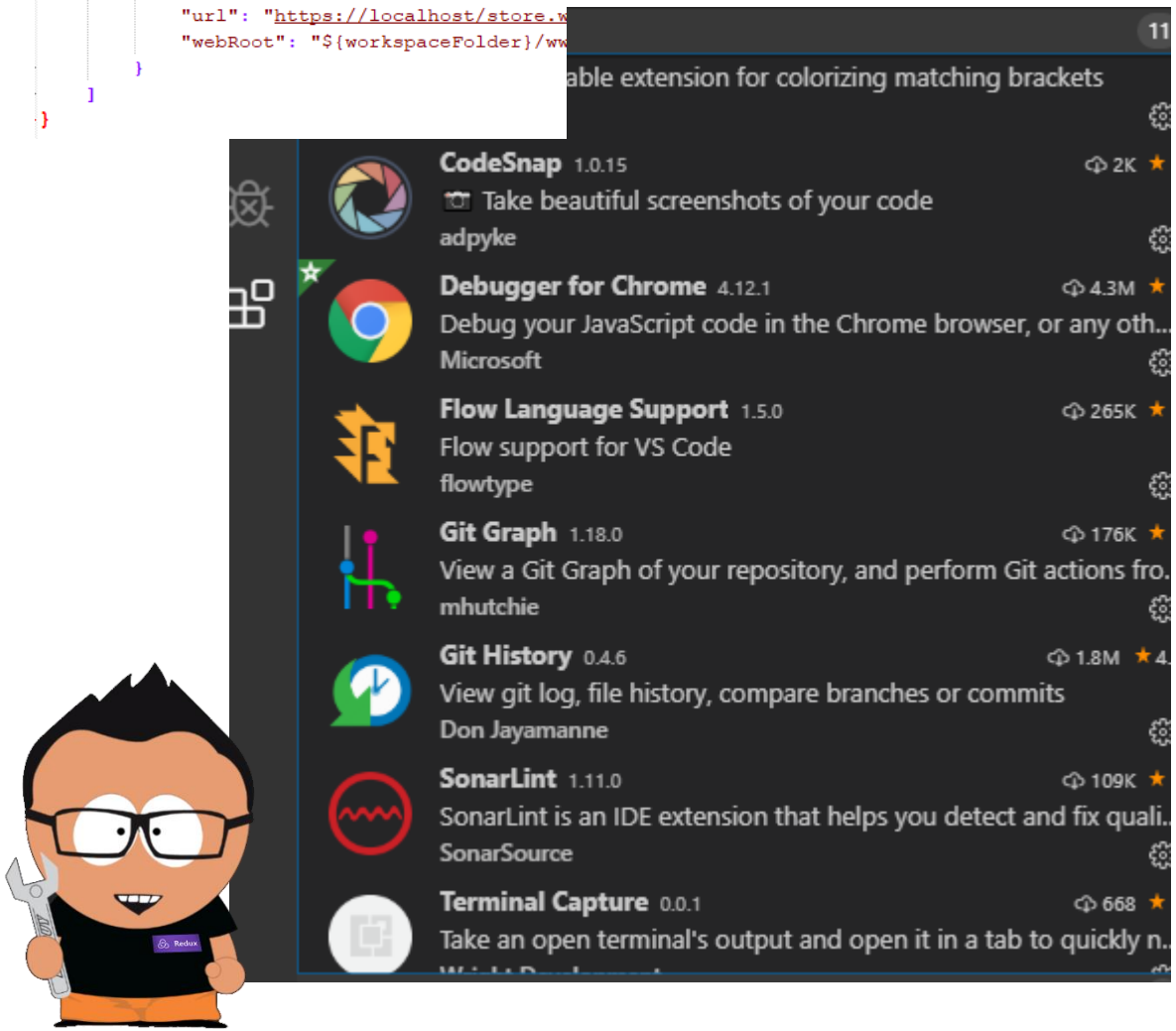
* le fichier .vscode/launch.json pour un projet (debug)
* ouvrir un terminal (cmd à l'intérieur de vscode)
* raccourci de commandes CTR + SHIFT+ P
* installer des extensions VisualStudio

Extension pour Chrome :

* DEV redux tool

Faire un compte sur codepen ou autre

```
{
    "version": "0.1.0",
    "configurations": [
        {
            "name": "launch debug localhost",
            "type": "chrome",
            "request": "launch",
            "url": "https://localhost/store.w
            "webRoot": "${workspaceFolder}/ww
        }
    ]
}
```



React-Redux

Redux

https://redux.js.org :

Inspiré de l'architecture Flux , **Redux** est une bibliothèque open source développée par **Dan Abramov** et **Andrew Clark** ,

pour répondre à cette problématique. Redux est une réécriture de l'architecture flux.

On la catégorise dans les « gestionnaire de State» (tel que MobX).

Indépendante de React, cette librarie est utilisée courrament avec react et angular. (+60% des applications React)

Dan Abramow : Redux, create-react-app

``You Might Not Need Redux`` : https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

Andrew Clark : Redux , FSA (Flux Standard Action)

``Redux is a stupid fucking event emitter with a disproportionately excellent ecosystem of tools built on top of it.

The good part of Redux is the reducers.The bad part is using a single immutable atom for the whole UI and

React-Redux

Redux adoption ?

- A-t-on besoin d'une couche supplémentaire dans les front dvi ?
  - Nous rencontrons des difficultés à structurer nos applications, la culture des composants react étant particulière
  - Nous avons besoin de séparer les responsabilités à notre niveau entre la partie purement UI et le reste (déporter les traitements des évènements de l'UI : handleClick => couche service ? Ou équivalente)
  - Nous avons souvent besoin de maintenir la cohérence du State au niveau applicatif (stratégie app-centric vs component-centric)
  - Nos types d'application grille/détail correspondent extrêmement bien au type d'application ayant besoin d'un outil de management de state
- Dans la catégorie gestionnaire de state , pourquoi plus Redux que MobX ?

https://codeburst.io/mobx-vs-redux-with-react-a-noobs-comparison-and-questions-382ba340be09

Redux et plus lourd mais permet de garder une maintenabilité :

- Respecte le principe d'immutablité :
- Un seul store
- Garde une data normalisée dans son store
- Meilleure scalabilité (adaptabilité à la montée en charge)
- Mieux adapté au travail en équipe
- Plus connu (mieux supporté par la communauté, plus de tutoriels,d'outils, d'aide , etc …)

React-Redux

Redux : fonctionnement et principes

« Redux c'est juste un gestionnaire d'évènement branché à un gestionnaire de variables...»

Redux permet la centralisation d'un ou plusieurs state dans un **Store**.

On va donc déporter la gestion du state des composants en les connectant directement au Store

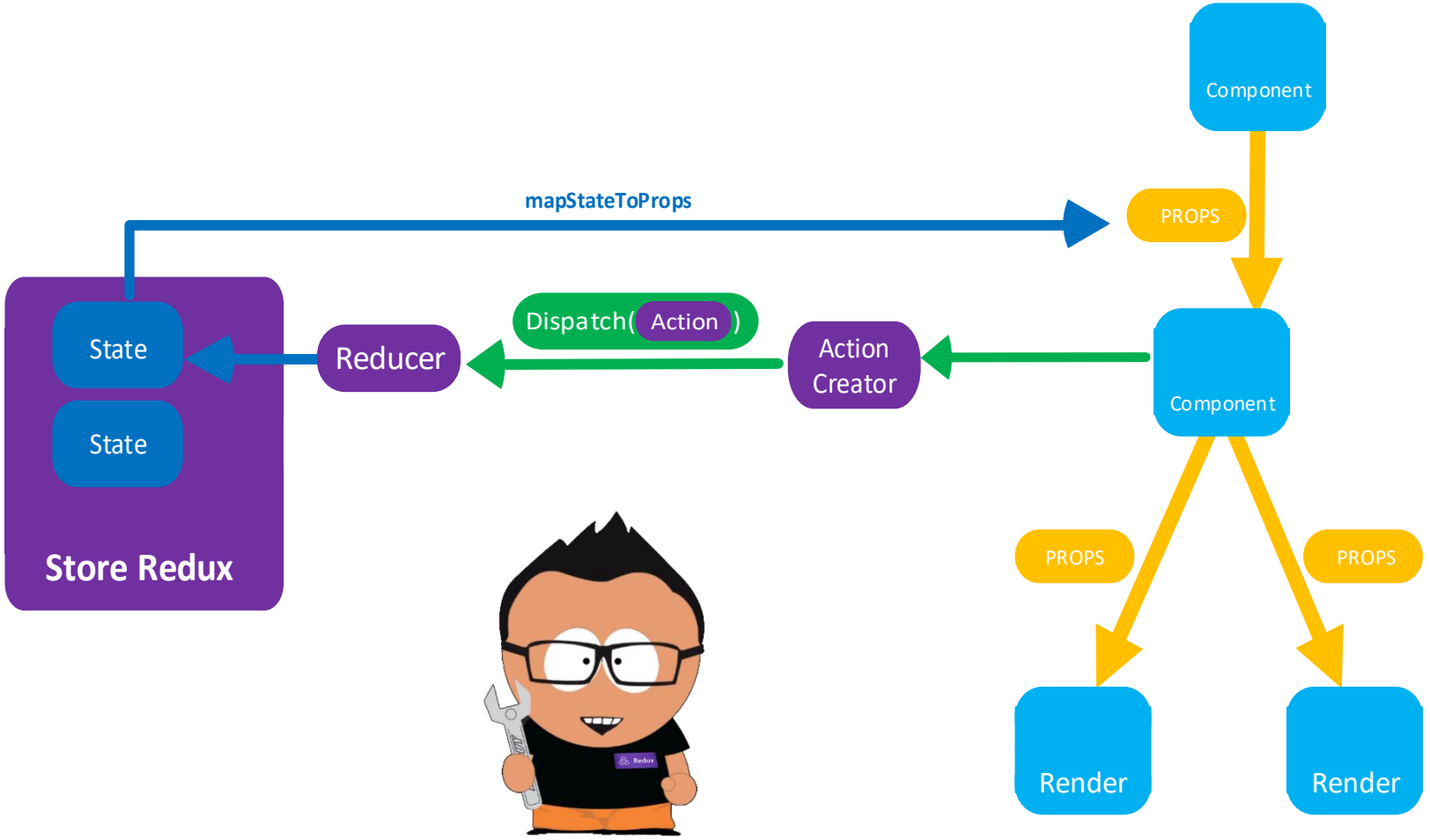
Et c'est tout.

- En lecture, seul le composant peut lire directement les valeurs du state.
- Pour mettre à jour le store, il faut envoyer un message, appelé **Action**, qui transporte (**dispatch**) les informations à destination du state.
- Une unité de travail, **Action creator** sera chargée de composer le message (action) et l'envoyer.
- Une autre unité de travail, le **reducer** sera chargé de réceptionner le message afin de produire une copie du state à destination du **Store**.

React-Redux

Redux : fonctionnement et principes

- *« Redux c'est juste un gestionnaire d'évènement branché à un gestionnaire de variables…»*



React-Redux

Redux

- Le Store :
  - le store est un objet global, avec quelques méthodes (utiles).
  - il contient un ou plusieurs state
  - seul un reducer peut demander à mettre à jour un state du store

Voici une représentation théorique d'une instance d'un store :

```
export const store =
{
    // exemple state provided by user
    mainState : {
        pageTitle : 'Item List',
        listItems : [...]
    },
    // another exemple state provided by user
    detailState : {
        pageTitle : 'detail'
        id : 2,
        itemTitle : 'Ipad',
        price : 1100,
        isNew : false
    },
    getState =() =>{/*...*/},                    // provided by redux lib
    dispatch  = (action) => {/*..*/},            // provided by redux lib
    subscribe = (listener)=> {/*..*/},           // provided by redux lib
    replaceReducer = (nextReducer) => {/*...*/}  // provided by redux lib
}
```

React-Redux

Redux :

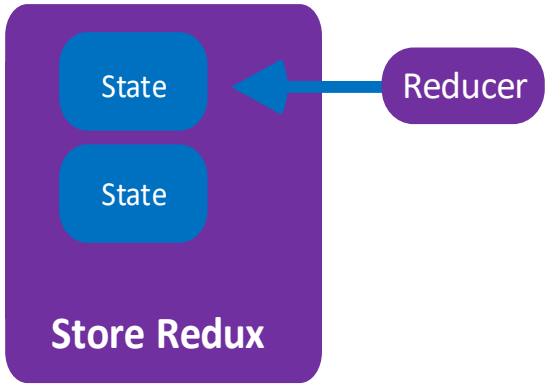
- le store est crée par le méthode createStore du module redux , il prend en paramètrer un reducer et un enhancer:

voici une implémentation minimale

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

const reducer = (state = [], action) => { return state; }

const store = createStore(reducer, ['Use Redux']);
```



Bien sûr, on va pouvoir décorer ce store

et brancher divers outils à traver l'enhancer , y compris un middleware (**Thunk**)

```
export const configureStore = () => {
    const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
    const store = createStore(
        createRootReducer(),
        composeEnhancers(
            applyMiddleware(
                thunk
            )
        )
    );
    return store;
}
```

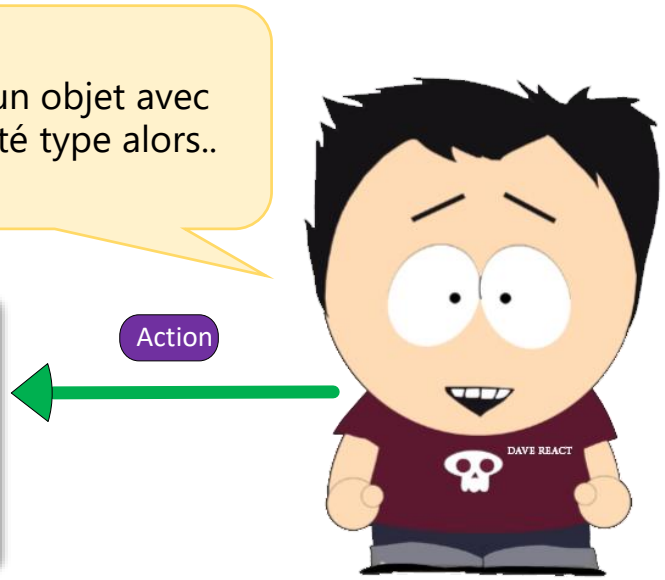
React-Redux

Redux

- **Les Actions** :
  - l'action est un message sous forme d'objet
  - il envoyé au reducer par un « action creator »
  - une action redux peut transporter un contenu :

```
const ReduxAction = {
    type: "TASK_COMPLETED",

    status: "completed"
    date: ...
    whatever :  ..
}
```



conceptuellement, on continue de le designer par le terme payload (en référence à flux).

En « flux », l'action est objet avec une clé sous forme de chaine de caractère et d'une propriété payload , un objet.

```
const FluxAction = {
    type: "TASK_COMPLETED",

    payload: {
        status: "completed",
        ...
    }
    metadata: {}
    error :  false
}
```

React-Redux

Redux : https://redux.js.org/basics/actions

Les Actions creator : l'action creator est l'intermediaire entre le composant react et le reducer. il envoie ou « dispatch » un action

Dans sa forme de base, **l'action creator** de Redux , crée et retourne une **action**.

Cela veux dire qu'elle ne modifie qu'une seule fois le state.

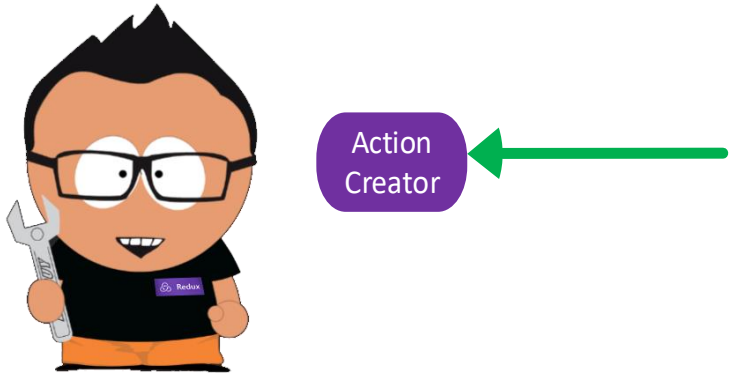
Il vous faudra après dispatcher cette action.

```
const addTodo = (text) => {
    return {
        type: ADD_TODO,
        text : text
    }
}
dispatch(addTodo(text))
```

Cependant avec le **Middleware thunk** (qu'on utilisera) **l'action creator** retournera **une fonction** ,

on pourra dispatcher autant de fois que l'on a envie et on pourra traiter les cas asynchrones.

```
const addTodo = (text) => {
    return async (dispatch, getState) => {
        /// do something :
        await Datamanager.getSomethingAsync();
        dispatch({
            type: ADD_TODO,
            text : text
        })
    }
}
```



React-Redux

Redux : https://redux.js.org/basics/actions

Le(s) Reducer(s)

Un reducer est une fonction qui prend en paramètre un State et une action,

Et qui renvoie un state.

Les nouvelles valeurs du state sont portées par l'action (payload).

```
const defaultState = {
    errors:{},
    popupTitle : 'Nouvelle catégorie',// : 'Modification de la catégorie',
    category : {"ID":-1, "TITLE": ''}
};

export default (state= defaultState,action) => {

    switch(action.type){

        case CATEGORY_INIT:
            return {
                ...state,
                popupTitle :  action.popupTitle,
                category :  action.category
            }

        case  CATEGORY_TITLE_CHANGED :
            return {
                ...state,
                category : action.category
            };
    }
    return state;
}
```

React-Redux

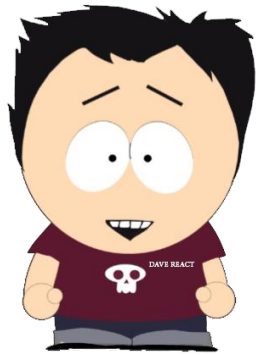
Redux : https://redux.js.org/faq/code-structure

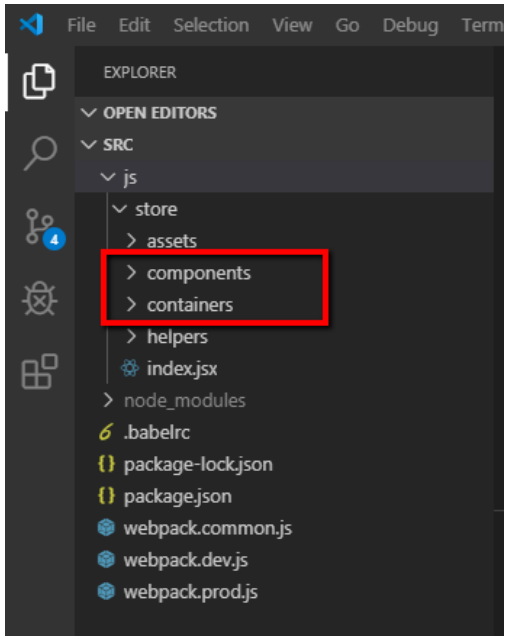
Architecture

Redux nous présente 3 approches courantes :

- **Rails-style**: separate folders for “actions”, “constants”, “reducers”, “containers”, and “components”
- **Domain-style**: separate folders per feature or domain, possibly with sub-folders per file type
- **“Ducks”:** similar to domain style, but explicitly tying together actions and reducers, often by defining them in the same file

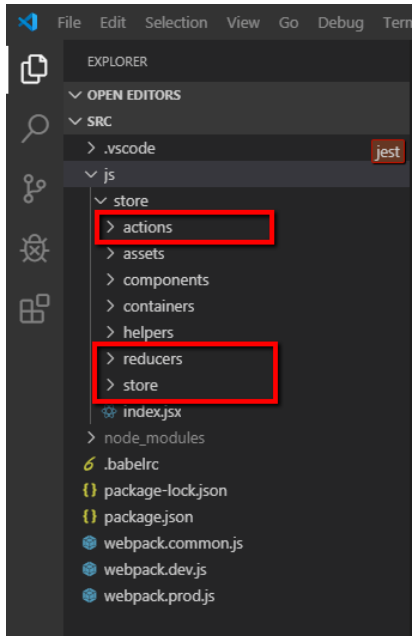
Ok , mais on met tout ça ou dans le projet ?





Voici un exemple avec le style rails (ruby and rails) Avant et après Redux.

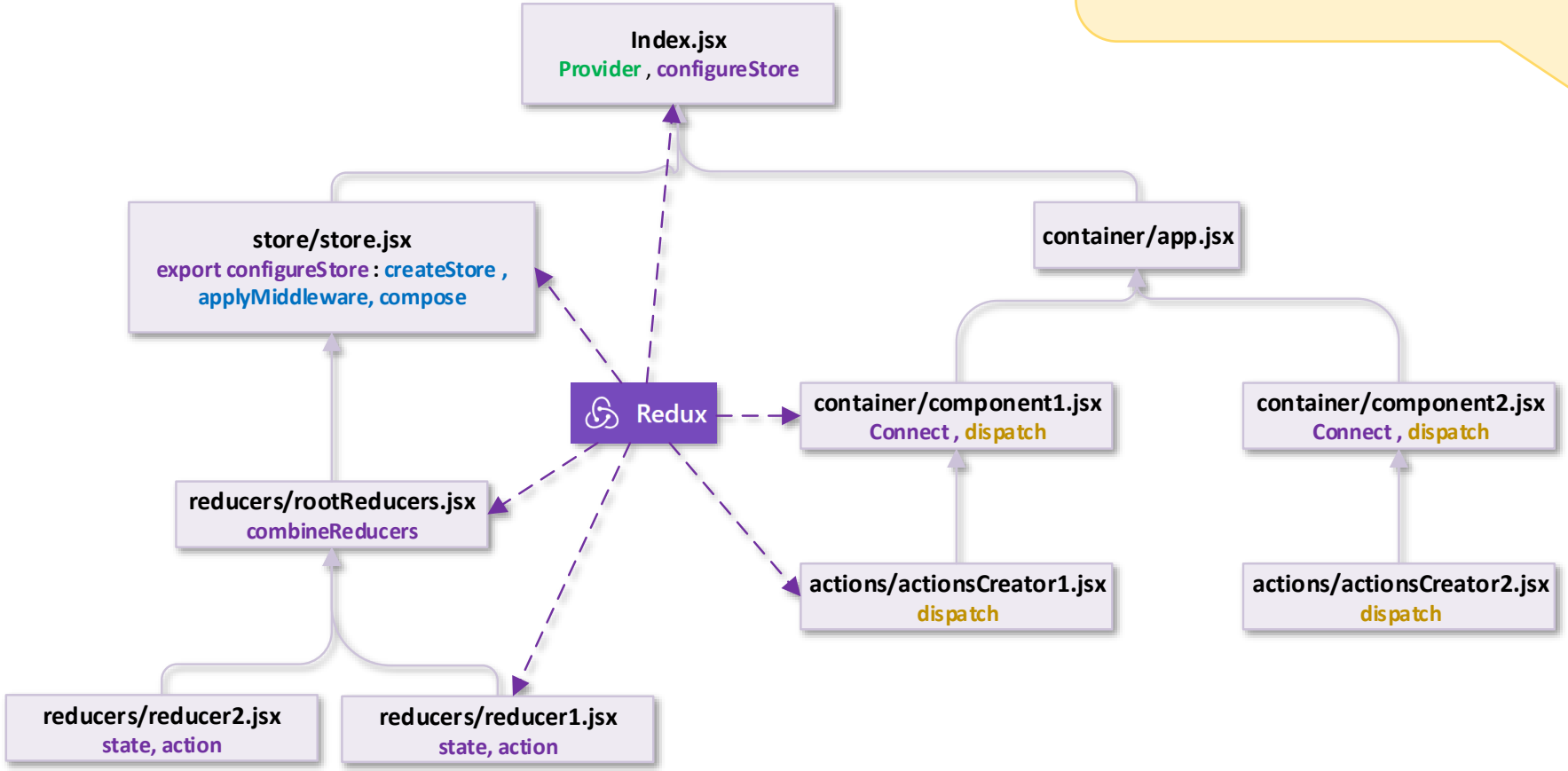
(le dossier au dessus « store» peut servir à separer les domaines)



React-Redux

- **Redux :** https://redux.js.org/faq/code-structure

- **Articulations**



Comment est ce qu'on organise tout ça ? (actionCreator, reducer, store) avec nos composant existants

React-Redux

Redux :

Implémentation

1, Connexion initialisation du Store : Nous allons utiliser 2 outils fournis par Re

- Dans le fichir index à la racine du projet - :

Après avoir initialisé le store avec **configureStore**, (cf ci-avant)

Provider encapsulera tous les composants qui pourront donc accéder au store

s'il sont connectés par le High Order Component **connect**.

```
// index.jsx
const store = configureStore();
ReactDOM.render(
    <Provider store={store}>
        <MyApp />
    </Provider>
    , reactContainer
);
```

Comment on initialise le store ?
Comment on le consulte ?
Comment on appel les actions creator ?

React-Redux

Redux :

Implémentation

2. Accès au Store depuis un Composant : mapStateToProps

Connect permet de transmettre une fonction à implémenter mapStateToProps renvoyant le state qui nous interesse

```
import { connect } from 'react-redux';

// implémentation :
class TodosList extend React.Component {
    render(){
        var { todosState } = this.props;
        return (
            <div>
                {todosState.todos.map((t, idx) => (<div key={idx} >{t.text}</div>) ) }
            </div>
        )
    }
}

// map state to props
const mapStateToProps = (state) => {
    return {
        todosState: state.todosState
    };
}

// wrap component with router, connect fetaures
export default connect(mapStateToProps, ... )(TodosList);
```

React-Redux

Redux :

Implémentation

2. Appeler un actionCreator depuis le composant

Le HOC connect permet à votre composant de beneficier le la fonction **dispatch** qui doit être utilisé pour appeler un méthode

```
import { connect } from 'react-redux';
import { addToDo }  from './actions/todoActions.jsx'

// implémentation :
class TodosList extend React.Component {

    addToDo = (todo) => {
        dispatch(addToDo(todo));
    }

    render(){
        var { todosState } = this.props;
        return (
            <div>
                {todosState.todos.map((t, idx) => (<div key={idx} >{t.text}</div>) ) }
            </div>
        )
    }
}

// map state to props
const mapStateToProps = (state) => {
    return {
        todosState: state.todosState
    };
}

// wrap component with router, connect fetaures
export default connect(mapStateToProps )(TodosList);
```

React-Redux

Redux :

Implémentation

2. Appeler un actionCreator depuis le composant : simplification : mapDispatchToProps et bind actionCreator

bindActionCreator prend en charge l'appel à **dispatch** pour simplifier l'appel d'une action

```
import { connect, bindActionCreators } from 'react-redux';
import { addToDo }  from './actions/todoActions.jsx'

// implémentation :
class TodosList extend React.Component {

    addToDo = (todo) => {
        this.props.addToDo(todo));
    }

    render(){
        var { todosState } = this.props;
        return (
            <div>
                {todosState.todos.map((t, idx) => (<div key={idx} >{t.text}</div>) ) }
            </div>
        )
    }
}

// map state to props
const mapStateToProps = (state) => {
    return {
        todosState: state.todosState
    };
}
// map dispatch to props
const mapDispatchToProps = (dispatch) => bindActionCreators({
    addToDo
}, dispatch);

// wrap component with router, connect fetaures
export default connect(mapStateToProps, mapDispatchToProps )(TodosList);
```

React-Redux

Redux : Middleware thunk

Thunk : https://github.com/reduxjs/redux-thunk

Le middleware apporte du confort à la programmation sous react-redux ,

Il permet surtour de gérer des appels asynchrone d'actions.

Il permet de retourner d'écrire des actions creator qui retournes des functions plutôt que des objets (actions)

(en référence au terme thunk)

Ref : What is Redux ? https://daveceddia.com/what-is-a-thunk/

Un **middleware** pour redux est une fonction en paramètre de la fonction **applyMiddleWare** qui elle-même peut être utilisé dans la fonction **createStore**

```
export const configureStore = () => {
    const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
    const store = createStore(
        createRootReducer(),
        composeEnhancers(
            applyMiddleware(
                thunk
            )
        )
    );
    return store;
}
```

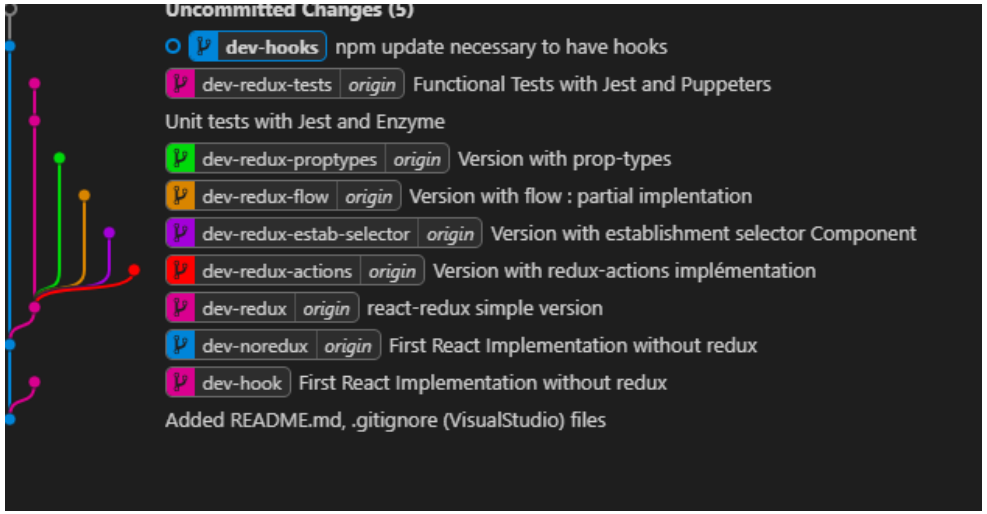
React-Redux

Redux :

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

Voici les branches de la démo:

- Version complète avec redux
- Version partielle avec redux-actions
- Version avec contrôle de typage prop-types
- Version avec contrôle de typage flow
- Version avec tests unitaires et tests fonctionnels
- Version avec hooks



React-Redux

Exercice - Redux : compléter

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

1. Aller sur la branche tp_redux

1. Examiner package.json , identifier les packages npm lié à redux
2. Compléter l'implémentation du store (store.jsx)
3. Importer et configurer Provider de 'react-redux' dans le fichier index.jsx
4. Aller dans le reducer ArticleListReducer et examiner le state d'initilisation , trouver la variable **isShowModalDialog**
5. Completer la méthode showdialog, dans actionsCommon.jsx, avec le type « **SHOW_DIALOG** » , completer le reducer
6. Completer la méthode closedialog, dans actionsCommon.jsx , avec le type « **CLOSE_DIALOG** » afin qu'elle soit « thenable » , completer le reducer
7. Aller dans ce container ArticleList , et connecter le à redux (connect) , examiner la méthode **mapStateToProps**
8. Toujours dans ArticleList , Importer la méthode **getArticles**, puis compléter le code dans l'évenement **handleClickRefresh et componentDidMount**, pour rafraichier la liste des articles avec **dispatch**, puis ecrire une seconde version en s'appuyant sur mapdispatchToProps
9. Examiner le fichier **rootReducer.jsx** ,compléter la fonction **combineReducers** pour prendre en compte le reducer immplémenté dans le fichier articleListReducer.jsx , (retourner dans le fichier ArticleList.jsx pour trouver le nom du state.)

React-Redux

Exemple composant avec State

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

1. Comparer l'implémentation **withReduxState** avec **withState**

1. Cela était utile ? Nécesssaire ? D'exporter le state vers redux ?

React-Redux

Review : transformer une application react en react-reux

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

Comparer les branche dev-no-redux et dev-redux.

Identifier les endroits ou la fonction setState est utilisée …

React-Redux

Variation d'écriture des actions et reducer avec redux-actions

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

branche dev-redux-actions

C'est nécessaire ? Pas facile à debugguer le reducer !

1. Bonne remarque !
2. Le retour vers Flux … bof
3. C'est conseillé dans certaine formations , déconseillé dans d'autres

React-Redux

Le typage avec prop-types : plus de contrôles

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

branche dev-redux-proptype

Whouah , j'ai pu voir une erreur qui passait inaperçue

1. C'est pas mal mais un peu leger…

React-Redux

Le typage avec Flow : plus de contrôles

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

branche dev-redux-flow

1. C'est plus contrraignat à écrire , il me trouve des erreurs, mais c'est souvent dans mon implémenation de flow…

1. Courage , avec un peu 'entrainement…

React-Redux

Les tests unitaires avec Jest et Enzyme

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

branche dev-redux-tests , aller sur le 1er commit

1. Faire du mock c'est pas si facile
2. Pour les test , j'ai compris c'est toujours :
3. **It('shoud be …') {**

   Expect(..)

   }

1. **JEST est très bien documenté** !! , beaucoups de produits reposent JEST plutôt que chai (même si chai et moka sont connus))
2. https://jestjs.io/docs/en/getting-started

React-Redux

Les tests fonctionnels (react/chrome uniquement) avec JEST et pupeeteer

Repository : https://domusvi.visualstudio.com/Portail%20Dvi/_git/HitechStoreDemo

branche dev-redux-tests, aller sur le 2ème commit

1. Pupeeteer c'est pour faire du pilotage de chrome, Jest prend la partie test, et il convient bien pour les fronts react encontinuité avec jest en test unitaire.
2. Il existe d'autres produits comme sélénium qui peuvent faire des tests fonctionnels (voire avec Alexandre)

React Hooks

Hooks : react >=16.8 initiation

référence: https://fr.reactjs.org/docs/hooks-intro.html

- branche dev-hooks

- Plus de classe , moins d'évenements

```
class Count extends React.Component {
    state = {
        count: 0
    };

    add = () => {
        this.setState({ count: this.state.count + 1 });
    }

    render() {
        return (
            <div>
                <p>Count: {this.state.count}</p>
                <button onClick={this.add}>Add</button>
            </div>
        );
    }
}
```

```
const Count = () => {
    // state variable, initialized to 0
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={() => setCount(count + 1)}>Add</button>
        </div>
    );
};
```

Est-ce vraiment un gain en nombre de ligne de code ?

Aujourd'hui le but est de savoir lire un code hook (exemple Scheduler Kendo)

+ branche dev-hooks

En savoir + sur le partage de state : partage de state

React render performance

Optimisation des Performances : shouldComponentUpdate, PureComponent

1. En résumé un PureComponent est un composant qui fait ça.
2. ça permet , dans des cas simple de laisser la main à react pour decider s'il doit mettre à jour le render du composant.
3. Par défaut shoulcomponentUpdate renvoie toujours true;
4. L'optimisation est souvent incompatible avec le pattern render props.

```
export class SampleComponent extends React.Component {
    shouldComponentUpdate(nextProps, nextState) {
        return shallowCompare(this, nextProps, nextState);
    }

    render() {
        return <div className={this.props.className}>foo</div>;
    }
}
```

```
export class SampleComponent extends React.PureComponent {
    render() {
        return <div className={this.props.className}>foo</div>;
    }
}
```

Purecomponent : **shallowCompare** et exemple non gérable par du shallowCompare : https://codepen.io/batran/pen/QWWGMQE

Version ou il faut un component pour optimiser : référence: https://codepen.io/batran/pen/QWWGMQE

React : Patterns avancés

HOC : High order Component , abandon des mixins

- Les mixins sont dangereux : https://fr.reactjs.org/blog/2016/07/13/mixins-considered-harmful.html
- Remplacé par les **HOC** : **High order component**
  - Ce sont des wrappers qui augmentent les capacités du composant wrappé :
  - exemple dans le repo Demo : **InputWithValidation !** Ici on veut **ajouter un texte en rouge** en dessous d'une Input quand il y a une erreur

Cf code page suivante :

* Le Principe simple et puissant , on pourra définir un alias via une const :

```
const InputWithValidator = withValidation(Input);


/// ...oqu'on utilisera comme ceci dans la partie render
    <InputWithValidator type="text" value={...} onChange={...} /></InputWithValidator >
/// ...
```

React : Patterns avancés

HOC : High order Component exemple

```
export default  (WrappedComponent) =>{

    class InputWithValidation extends React.Component {

        constructor(props) {
            super(props);
        }

        render(){
            var newClassName = this.props.className||'';
            if (this.props.errors){
                if(this.props.errors[this.props.name]){
                    newClassName += " app-input-valid-error";
                }
            }

            const {forwardedRef, ...rest} = this.props;
            return(
                <div>
                    <WrappedComponent  ref={this.props.forwardedRef}  {...rest} className={newClassName}  />
                    {this.props.errors[this.props.name] &&
                     (<div style={{ color:'red'}}>
                            <span className="fa fa-exclamation-circle"> </span>
                            <span style={{marginLeft:'8px'}} >{ this.props.errors[this.props.name]  }</span></div>)
                    }
                </div>
            );
        }
    }

    return React.forwardRef((props, ref) => {
        return <InputWithValidation {...props} forwardedRef={ref} />;
    });
}
```

React : Patterns avancés

HOC : High order Component : forward de ref

Quand vous utilisez **{...props}** , vous récupérer toutes les props du parent pour les transmettre à l'enfant, cependant **ref est une exception** et n'est pas une props transmise.

Par défaut ref pointera sur le wrapper et non l'objet wrappé.

Pour transmettre une ref au composant wrappé

Il faut utiliser **React.forwardRef**

```
export default  (WrappedComponent) =>{

    class InputWithValidation extends React.Component {

            const {forwardedRef, ...rest} = this.props;
            return(
                <div>
                    <WrappedComponent  ref={this.props.forwardedRef}  {...rest} className={newClassName}  />
                </div>
            );
        }
    }

    return React.forwardRef((props, ref) => {
        return <InputWithValidation {...props} forwardedRef={ref} />;
    });
}
```

React : Patterns avancés

- **Fragment :** https://fr.reactjs.org/docs/fragments.html

Fragments
En React, il est courant pour un composant de renvoyer plusieurs éléments. Les fragments nous permettent de grouper une liste d'enfants sans ajouter de nœud supplémentaire au DOM.

```
render() {
    return (
        <React.Fragment>
            <ChildA />
            <ChildB />
            <ChildC />
        </React.Fragment>
    );
}
```

React : Patterns avancés

Render Props : propriété de rendu

https://fr.reactjs.org/docs/render-props.html

1. Technique pour les developpeurs de composants

2. En react-hook le patern n''existe plus .remplacé par le principe du hook

```
class Mouse extends React.Component {
    constructor(props) {
        super(props);
        this.handleMouseMove = this.handleMouseMove.bind(this);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove(event) {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div style={{ height: '100%' }} onMouseMove={this.handleMouseMove}>

                {/*
          Au lieu de fournir une représentation statique de ce qu'affiche <Mouse>,
          utilisez la prop `render` pour déterminer dynamiquement quoi afficher.
        */}
                {this.props.render(this.state)}
            </div>
        );
    }
}

class MouseTracker extends React.Component {
    render() {
        return (
            <div>
                <h1>Déplacez votre souris sur l'écran !</h1>
                <Mouse render={mouse => (
                    <Cat mouse={mouse} />
                )} />
            </div>
        );
    }
}
```

React : Patterns avancés

Portals

https://fr.reactjs.org/docs/portals.html

```
render() {
    // React *ne crée pas* une nouvelle div, mais affiche les enfants dans `domNode`.
    // `domNode` peut être n'importe quel élément valide du DOM, peu importe sa position.
    return ReactDOM.createPortal(
        this.props.children,
        domNode
    );
}
```

React peut gérer un « Portal » pour gérer en affichage en dehors de son nœud parent principal :

https://codepen.io/batran/pen/bGGmeMp

React animation

Animation : react Motion , popmotion , cssTransition

react-motion

La librairie d'animation a été faite avec le principe des render props :

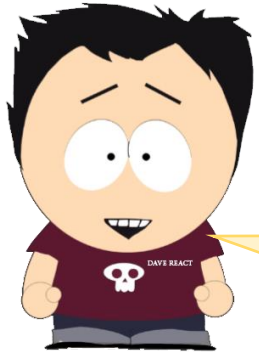
https://github.com/chenglou/react-motion

Sinon pour faire tout en react/css il y a **csstransition**

http://reactcommunity.org/react-transition-group/css-transition

Popmotion

https://popmotion.io/pose/examples/svg-morphing/



1. Moi j'aime bien popmotion
2. Sinon en pure css https://daneden.github.io/animate.css/

React notions avancées

Immutabilité

Un state doit être immutable (on ne pas changer directement ses valeurs) , car entrainerait des desynchronisation entre le DOM virtuel et le DOM réel)= bugs inexplcables

Beaucoup de solution de copy (object , assign,) se sont averées inefficaces dans nos DEV.

Attention , le spread opertor souvent présenté comme la solution , ne fait que de la s**hallow copy** .. (copie de surface)

Pour etre spur de faire de la copie intégrale et non de la copie de référence, nous avons utilisé la librairie **deep-clone.**

Facebook a créé la librairie ImmutableJS forçant l'utilisation de getter et setter (old style C#/java) pour gérerer les List, Map et Set , mais c'est la croix la banière à utiliser , je vous le deconseille,..

car on a tendance à melanger les type immutable avec les type non immutables dans le code….

React notions avancées

Immutabilité

Un state doit être immutable (on ne pas changer directement ses valeurs) , car entrainerait des desynchronisation entre le DOM virtuel et le DOM réel)= bugs inexplcables

Beaucoup de solution de copy (object , assign,) se sont averées inefficaces dans nos DEV.

Attention , le spread opertor souvent présenté comme la solution , ne fait que de la s**hallow copy** .. (copie de surface)

Pour etre spur de faire de la copie intégrale et non de la copie de référence, nous avons utilisé la librairie **deep-clone.**

Facebook a créé la librairie ImmutableJS forçant l'utilisation de getter et setter (old style C#/java) pour gérerer les List, Map et Set , mais c'est la croix la banière à utiliser , je vous le deconseille,..

car on a tendance à melanger les type immutable avec les type non immutables dans le code….