
React

l'essentiel

Table des matières

I - React (présentation).....	3
1. Présentation de React.....	3
II - JSX et structure des composants de React.....	7
1. Structure d'un composant react.....	7
III - Router react (navigations).....	17
1. Routing avec react.....	17
IV - Appel api rest depuis react.....	19
1. Appel d'api REST depuis react.....	19
V - Redux.....	20

1. L'essentiel de redux.....	20
2. Utilisation de redux au sein de react.....	24

VI - Annexe – rappels javascript / es 2015.....	31
--	-----------

1. Mots clefs "let" et "const" (es2015).....	31
2. "Arrow function" et "lexical this" (es2015).....	32
3. for...of (es2015) utilisant itérateurs internes.....	33
4. Prog. orientée objet "es2015" (class, extends, ...)......	38
5. Modules (es2015).....	46
6. Promise (es2015).....	54

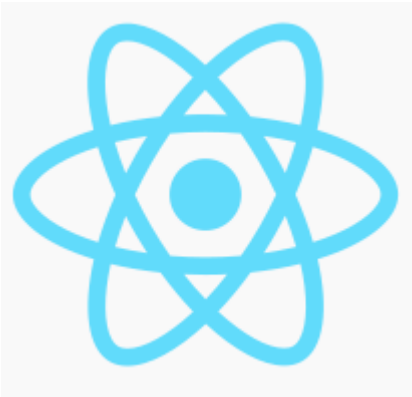
VII - Annexe – Bibliographie, Liens WEB + TP.....	63
--	-----------

1. Bibliographie et liens vers sites "internet".....	63
2. TP.....	63

I - React (présentation)

1. Présentation de React

1.1. Principales fonctionnalités de Réact



React (alias **React.js** ou **ReactJS**) est une **bibliothèque** JavaScript **libre** développée par **Facebook** depuis 2013.

Cette librairie (ou petit **framework**) sert à **développer des applications "web"** en mode **SPA** (*Single page application*) avec une seule page html constituée par une hiérarchie de composants dont certains sont interchangeables .

Chaque composant à un rendu (portion html) dépendant d'un état et réactualisé à chaque changement d'état (selon événements ou autres) .

Principe de React au niveau de l'arborescence des composants :
les **données descendent** (vers les composants "fils") et l'**état remonte** .

Les principaux concurrents du framework "*React*" sont "*Angular*" et "*VueJs*" .

"React" est un framework plus **léger** que "*Angular*" qui se concentre sur l'aspect "Visualisation" avec une logique "MVC coté navigateur/client" .

Au niveau d'un composant, contrairement à angular qui distingue bien "template html" et "code javascript/typescript" en arrière plan , le **framework "React" a tendance à tout fondre en un seul bloc de code (fichier .jsx comportant à la fois html et javascript)** .

Le framework "**React**" s'appuie en interne sur un "**DOM Virtuel**" **entièrement en javascript** .

Il existe une version dérivée appelée "**react native**" pour le développement d'applications mobiles (android, iphone, ...) .

1.2. "Hello world" avec React en mode "cdn"

Exemple inspiré du tutoriel officiel "<https://www.taniarascia.com/getting-started-with-react/>"

index.html

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Hello World with React</title>
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
</head>

<body>
  <div id="root"></div>

  <script type="text/babel">
    class App extends React.Component {
      render() {
        return <h1>Hello world!</h1>
          <!-- JSX syntax (html with js) but not return string -->
        }
      }

    ReactDOM.render(<App />, document.getElementById('root'))
  </script>
</body>
</html>
```

NB : cet exemple basique directement basé sur un téléchargement cdn et une transpilation es6/es2015 -> es5 via babel en mode dynamique n'est pas du tout optimisé pour la production mais a le mérite de montrer le code minimum d'une application react .

NB : babel.js (et type="text/babel") est également utilisé pour interpréter la syntaxe JSX .

1.3. Utilitaire "create-react-app" (CRA)

CRA (*create-react-app*) est un petit programme utilitaire (facultatif mais très pratique) en ligne de commande permettant de créer rapidement un début d'application react .

```
npm install -g create-react-app
```

```
cd ...
```

```
create-react-app my-react-app
```

Arborescence construite :

<ul style="list-style-type: none"> node_modules public src .gitignore package.json package-lock.json README.md 	<p>dans src :</p> <ul style="list-style-type: none"> App.css App.js App.test.js index.css index.js logo.svg serviceWorker.js setupTests.js 	<p>dans public :</p> <ul style="list-style-type: none"> favicon.ico index.html logo192.png logo512.png manifest.json robots.txt
---	--	---

package.json

```
{
  "name": "my-react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  ...
}
```

cd my-react-app

npm start (ou npm run start) --> **lance un serveur de test** (*http://localhost:3000/*)

npm run build --> construit (dans répertoire "build")
une version "avec bundle js" pour la production

NB : le code construit dans build **nécessite http pour fonctionner** (lancer au minimum lite-server dans build après une installation via npm install -g lite-server)

npm test (ou npm run test) --> *lance des tests unitaires*

code minimaliste dans **public/index.html**

```
<body>
  <div id="root"></div>
</body>
```

code minimaliste dans **src/index.css**

```
h1 { color: blue; }
.App { font-style : italic;}
```

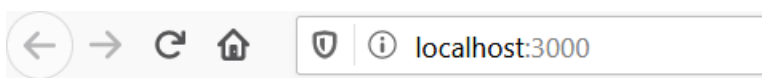
code minimaliste dans **src/index.js**

```
import React , { Component } from 'react';
import ReactDOM from 'react-dom';
import './index.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello world with React</h1>
      </div>
    )
  }
}

ReactDOM.render(<App />, document.getElementById('root'))
```

Résultat (avec npm start) :



Hello world with React

II - JSX et structure des composants de React

1. Structure d'un composant react

1.1. JSX (Js + Xml)

Sans jsx (en pur javascript) :

```
class App extends Component {
  render() {
    const heading = React.createElement('h1', { className: 'App' },
'Hello world React without jsx')
    return heading;
  }
}
```

Avec jsx :

```
class App extends Component {
  render() {
    const heading = <h1 className="App">Hello world React with jsx</h1>
    return heading;
  }
}
```

NB1: La syntaxe JSX (js + xml) est retransformée en interne en pur js .

En jsx, le mot clef class est réservé et donc className="cssClassName" plutôt que class="..."

et toutes les règles de syntaxe XML (balise bien fermée)

et camelCase pour nom de attribut et méthode (ex : onClick="..." plutôt que onclick="..")

NB : en syntaxe JSX , le remplacement d'une variable javascript par son contenu s'effectue via en entourage de celle-ci par des accolades :

```
class MyHeader extends Component{
  render() {
    const headerText = 'myHeader default text';
    return <div> {headerText} </div>
  }
}
```

rendu -->

myHeader default text

Pour éviter des conflits avec des mots réservés du langage javascript , quelques adaptations en JSX, avec par exemple `htmlFor="idInput"` à la place de `for="idInput"` :

```
<label htmlFor="montant">montant:</label>
```

```
<input id="montant" type="text" .... />
```


1.2. "class component" & "simple/lambda component"

```

...
//class Component
class MyHeader extends Component{
  render() {
    return <div> myHeader </div>
  }
}
//NB : pas besoin de () si return en une seule ligne

//Simple component as arrow function :
const MyFooter = () => {
  return <div> myHeader </div>
}

class App extends Component {
  render() {
    return (
      <div>
        <MyHeader />
        <h1 className="App">Hello world React</h1>
        <MyFooter />
      </div>
    )
  }
}
//NB : besoin de () et d'une div englobante pour intégrer plusieurs sous composants adjacents .
...

```

Chaque sous-composant "react" est vu comme une balise xml/jsx .

1.3. Passage de propriétés (props) à un sous composant

Exemple:

my-table.js

```
import React, { Component } from 'react';

export class MyTable extends Component {
  render() {
    const { title, peoples } = this.props;
    //ou bien const peoples = this.props['peoples']; ...
    //ou bien const peoples = this.props.peoples; ...
    const peoplesJsonString = JSON.stringify(peoples);
    return <div> {title} myTable {peoplesJsonString} </div>
  }
};
```

NB : *this.props* permet de récupérer toutes les valeurs des propriétés passées à un sous composant.

index.js

```
...
import { MyTable } from './my-table';
...
class App extends Component {
  constructor() {
    super();
    this.appPeoples = [ { firstName : 'jean' , lastName : 'Bon' } ,
                        { firstName : 'alain' , lastName : 'Therieur' } ];
  }

  render() {
    return (
      <div>...
        <MyTable title="myTitle" peoples={this.appPeoples} />
      ... </div>
    )
  }
}
...

```

NB : on peut utiliser n'importe quel nom de propriétés (en attribut de la balise du sous composant) tant que ce n'est pas un mot réservé du langage .

1.4. boucle sur éléments d'un tableau en JSX

```
...
export class MyTable extends Component {
  render() {
    const { title, peoples } = this.props;
    const peoplesRows = peoples.map((row, index) => {
    return (
      <tr key={index}>
        <td>{row.firstName}</td>
        <td>{row.lastName}</td>
      </tr>
    );
    });

    return (
      <div>
        <h3> {title} </h3>
        <table border='1'>
          <thead>
            <tr><th>firstName</th><th>lastName</th></tr>
          </thead>
          <tbody>
            {peoplesRows}
          </tbody>
        </table>
      </div>
    );
  }
};
```

rendu :

firstName	lastName
jean	Bon
alain	Therieur

1.5. state & setState()

Un composant "react" est conçu pour se rafraîchir visuellement dès qu'il change d'état (**this.state**). La méthode asynchrone **setState(...)** héritée de **React.Component** permet de modifier une partie de l'état . Le second paramètre facultatif de **setState()** est une callback appelée quand la mise à jour aura été effectuée au niveau de **this.state** .

Exemple :

```
import React, { Component } from 'react';

export class MyTva extends Component {
  constructor(props) {
    super(props);
    this.state = { ht: 0, taux:20, tva:0, ttc : 0 };
  }

  recalculerTvaEtTtc(){
    this.setState({ tva : Number(this.state.ht * this.state.taux / 100) ,
                  ttc : Number(this.state.ht * (1 + this.state.taux/100)) });
  }

  handleChange(keyName,evt) {
    //setState() hérité de Component modifie qu'une partie de l'état en mode asynchrone

    //this.setState({[keyName]: evt.target.value}); //attention : asynchrone , pas immédiat !!!

    //le second paramètre facultatif de setState() est une callback appelée qd maj effectuée
    this.setState({[keyName]: evt.target.value} ,
      ()=>{ console.log("state (before recalcul)=" + JSON.stringify(this.state));
            this.recalculerTvaEtTtc();
          });
  }

  render() {
    return (<div>
      ht: <input type="text" value={this.state.ht}
                onChange={(e)=>this.handleChange('ht',e)} /> <br/>
      taux(%): <input type="text" value={this.state.taux}
                onChange={(e)=>this.handleChange('taux',e)} /> <br/>
      {this.state.tva!=0 ? (
        <span>tva: {this.state.tva} , ttc: {this.state.ttc}</span>
      ) : ( <span><i>pas de tva</i></span> )
      }
    </div>
  );
  };
};
```

ht:

taux(%):

tva: 40 , ttc: 240

1.6. remontée d'événement d'un composant à son parent

this.props peut comporter des références sur des fonctions événementielles du parent.

Exemple :

people list

firstName	lastName	action
jean	Bon	<input type="button" value="delete"/>
alain	Therieur	<input type="button" value="delete"/>

my-data-table.js

```
import React, { Component } from 'react';

export class MyDataTable extends Component {
  render() {
    const { data, columnNames } = this.props;

    const thList = columnNames.map((colName, index) => {
      return ( <th key={index}>{colName}</th> );
    });
    const headerRow = <tr>{thList}<th>action</th></tr>;

    const dataRows = data.map((row, index) => {
      const tdList = [];
      for (const [index, colName] of columnNames.entries()) {
        tdList.push(<td key={index}>{row[colName]}</td>)
      }
      return (
        <tr key={index} >
          {tdList}
          <td>
            <input type="button" value="delete"
              onClick={() => this.props.onDelete(index)} />
          </td>
        </tr>);
    });

    return (
      <div>
        <table border='1'>
          <thead>{headerRow}</thead>
          <tbody>{dataRows}</tbody>
        </table>
      </div>
    );
  }
}
```

people.js

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import { MyDataTable } from './my-data-table';

export class PeopleComponent extends Component {
  constructor() {
    super();
    this.state = {
      peopleFields : [ 'firstName', 'lastName' ],
      peoples : [ { firstName : 'jean', lastName : 'Bon' },
                    { firstName : 'alain', lastName : 'Therieur' },
                    { firstName : 'alex', lastName : 'Therieur' },
                    { firstName : 'axelle', lastName : 'Aire' } ]
    };
  }

  onDeletePeople(index){
    const peoples = this.state.peoples;
    this.setState({ peoples: peoples.filter((p, i) => { return i !== index }) });
  }

  render() {
    return (
      <div>
        <h3 className="App">people list</h3>
        <MyDataTable columnNames={this.state.peopleFields}
                      data={this.state.peoples}
                      onDelete={(index)=>{this.onDeletePeople(index);}} />
      </div>
    )
  }
}
```

1.7. intégration de bootstrap-css et icons dans react

```
npm install --save bootstrap
```

dans **index.js**

```
...
import 'bootstrap/dist/css/bootstrap.css';
// Put any other imports below so that CSS from your
// components takes precedence over default styles.
import './index.css';
...
```

```
npm install --save react-icons
```

xyz.js

```
...
import { FaAngleDown, FaAngleUp } from 'react-icons/fa';
...
render() {
  const downOrUpIcon = this.state.toggleP ? <FaAngleUp /> : <FaAngleDown />;
  return ( <div>
    {downOrUpIcon}
  </div> );
}
...
```

documentation complète sur <https://react-icons.netlify.com/#/>

1.8. prop.children et imbrication de sous composants

panel1 ▼

panel1 ▲

contenu panneau 1

home.js

```
import React from 'react'
import {MyTogglePanel} from './my-toggle-panel'

class Home extends React.Component {
  render() {
    return (
      <div>
        <h1>Welcome to my react app (home)</h1>
        <MyTogglePanel title="panel1">
          <div>contenu panneau 1</div>
        </MyTogglePanel>

        <MyTogglePanel title="panel2">
          <div>contenu panneau 2</div>
        </MyTogglePanel>
      </div> )
    }
  }
}
export default Home
```

my-toggle-panel.js

```
import React, { Component } from 'react';
import { FaAngleDown, FaAngleUp } from 'react-icons/fa';

export class MyTogglePanel extends Component{

  constructor(props){ super(props); this.state = { toggleP : false };
  }

  toggleThisPanel(){ this.setState( { toggleP : !this.state.toggleP });
  }

  render() {
    const cardBodyCollapseClasses = "card-body collapse";
    const withShowClassOrNot =
      this.state.toggleP?cardBodyCollapseClasses+" show":cardBodyCollapseClasses;
    const downOrUpIcon = this.state.toggleP?<FaAngleUp/>:<FaAngleDown/>;

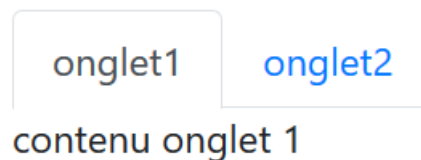
    return (
      <div className="card">
        <h4 className="card-header bg-secondary">
          <a className="text-light" onClick={()=>this.toggleThisPanel()} >{this.props.title}
            {downOrUpIcon} </a>
        </h4>
        <div className={withShowClassOrNot}>
          {this.props.children}
        </div>
      </div> );
  }
};
```

1.9. react-bootstrap (composants prédéfinis "bootstrap")

<https://react-bootstrap.netlify.com/>

```
npm install --save bootstrap
npm install react-bootstrap --save
```

```
import { Tabs, Tab } from 'react-bootstrap';
...
<Tabs defaultActiveKey="o1" >
  <Tab eventKey="o1" title="onglet1">
    <div>contenu onglet 1</div>
  </Tab>
  <Tab eventKey="o2" title="onglet2">
    <div>contenu onglet 2</div>
  </Tab>
</Tabs>
```



1. Routing avec react

1.1. Utilisation élémentaire de Router , Route et Link

```
import React , { Component } from 'react';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'

....

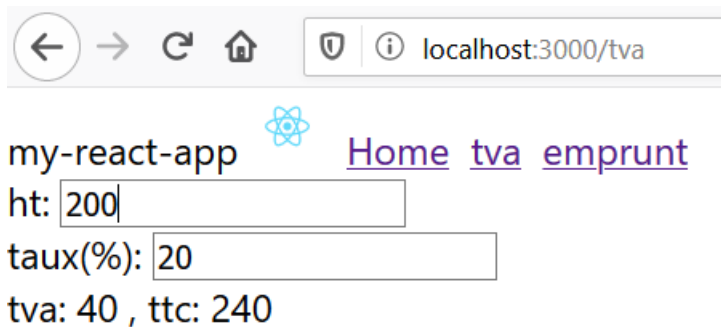
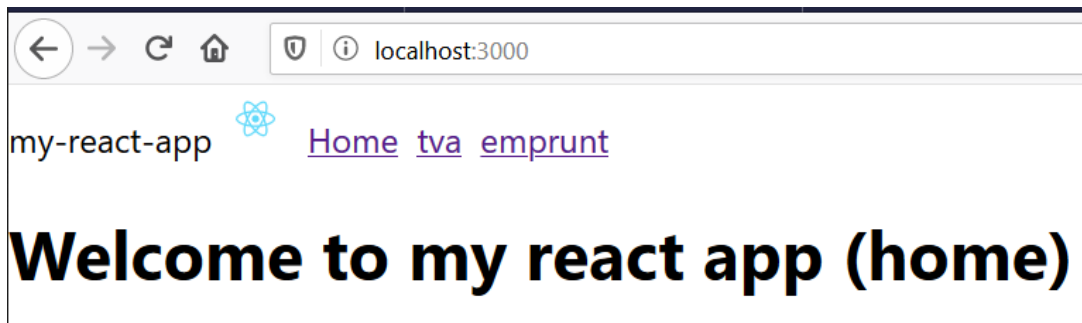
class MyHeader extends Component{
  render() {
    return (<header className="App-header"> my-react-app &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~</div>
</div>
)

const MyRoutingZone = () => {
  return (<div>
    <Route exact path="/" component={Home} />
    <Route path="/tva" component={MyTva} />
    <Route path="/emprunt" component={MyEmprunt} />
  </div>)
}

/* NB: <Link> and <Route> must be nested (indirectly) in <Router> */

function App() {
  return ( <div className="App">
    <Router>
      <MyHeader />
      <MyRoutingZone />
    </Router>
  </div> );
}

export default App;
```



1.2. Url pour Approfondir du Router de react

<https://codeburst.io/getting-started-with-react-router-5c978f70df91>

IV - Appel api rest depuis react

1. Appel d'api REST depuis react

1.1. Exemple

devises.js

```
import React from 'react'
import {MyTogglePanel} from './my-toggle-panel'

export class DevisesComponent extends React.Component {
  state = { data: [] }

  // Code is invoked after the component is mounted/inserted into the DOM tree.
  componentDidMount() {
    const url = 'http://localhost:8282/devise-api/public/devise'
    //with cross origin authorized access and served by tp_node_js/backend-tp-api
    fetch(url)
      .then(responseResult => responseResult.json())
      .then(resultObject => {
        console.log(JSON.stringify(resultObject));
        this.setState({
          data: resultObject,
        })
      })
  }

  render() {
    const { data } = this.state;
    const liList = data.map((entry, index) => {
      //const deviseAsString = JSON.stringify(entry);
      //return <li key={index}>{deviseAsString}</li>
      return <li key={index}> [{entry.code}] {entry.name} {entry.change}</li>
    })

    return ( <div>
      <h3>devises list (fetch from rest api)</h3>
      <ul>{liList}</ul>
    </div> )
  }
}
```

devises list (fetch from rest api)

- [EUR] Euro 0.92
- [GBP] Livre 0.82
- [JPY] Yen 132.02
- [USD] Dollar 1

V - Redux

1. L'essentiel de redux

1.1. Utilité de redux

Avec react sans redux l'état d'une application est complètement décentralisé à travers tous les composants de celle-ci et les mises à jour des states des principaux composants ne suivent pas d'ordre précis à cause du temps de réponse du serveur côté API, de l'asynchronicité, de la gestion des routes, etc.

Redux (version simplifiée de "flux") sert à **maintenir**, en arrière plan des "vues" de react, **un état global** à l'application bien cohérent et bien synchronisé entre tous les composants .

1.2. Concepts et terminologie de redux

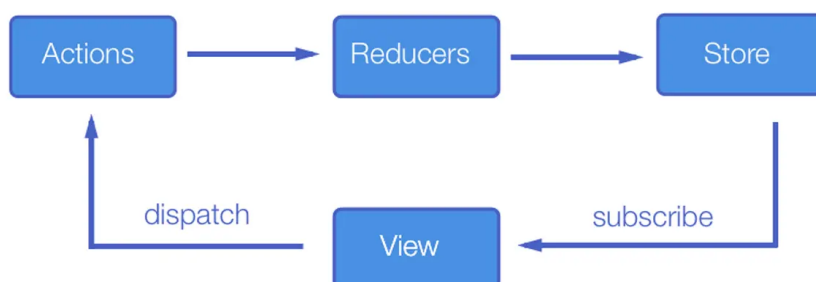
Éléments de redux	Rôles/caractéristiques
global state	objet principal (permettant de mémoriser l'état de toute l'appli) comportant de multiples sous objets <i>immutables</i> et <i>remplacés</i> .
action	objet " <i>demande de traitement</i> " permettant de modifier le "global state" (proche d'un objet " <i>command</i> " du design pattern de même nom)
reducer	objet technique sous forme de fonction pure permettant d'appliquer une action sur tout ou une partie d'un "global state existant" de façon à générer un "nouveau global state" ou une nouvelle sous-partie.
store	là où est stocké le "global state"

Le terme "*reducer*" vient du type de fonction que l'on peut appliquer à [Array.prototype.reduce\(\)](#).

Principales propriétés du global state :

- unique (mais composé de plein de substates complémentaires : un par "reducer")
- en lecture seule (mais dont certaines parties sont remplacée par de nouvelles)

Redux



1.3. Intégration et utilisation élémentaire de redux

npm install --save redux
npm install --save react-redux

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore } from 'redux'
import { Provider } from 'react-redux'
import 'bootstrap/dist/css/bootstrap.css'; import './index.css';
import App from './App';
import { myRootReducer } from './myCombinedRootReducer' ;
import { createIncrementCounterAction ,
        createDecrementCounterAction } from './counterActions' ;

// Create a Redux store holding the global state of your app.
// Its API is { subscribe, dispatch, getState }.
const store = createStore(myRootReducer);

//NB: all nested reducers in combined "myRootReducer"
// will be automatically called here
// to initialize each "substate" with a @@redux/INIT... action
// and with default value of (sub)state .

console.log("**initial global state="+JSON.stringify(store.getState()));

const unsubscribeLog = store.subscribe(() => console.log(">>global
state="+JSON.stringify(store.getState())))

// The only way to mutate the internal state is to dispatch an action.
// The actions can be serialized, logged or stored and later replayed.
store.dispatch(createIncrementCounterAction());
store.dispatch(createDecrementCounterAction());

unsubscribeLog();

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

NB : dans cet exemple , la partie grisée correspond à des petits tests (pour apprendre) , le reste correspond à la structure type d'une véritable appli "react + redux" .

actionsTypesConstants.js

```
export const INCREMENT_COUNTER = 'INCREMENT_COUNTER'
export const DECREMENT_COUNTER = 'DECREMENT_COUNTER'
....
```

counterActions.js

```
import * as types from './actionsTypesConstants'
//this file contains "actions creators" functions:
//each "created actions" is sort of "command/action description":

export const createIncrementCounterAction = () => ({
  type: types.INCREMENT_COUNTER
})

export const createDecrementCounterAction = () => ({
  type: types.DECREMENT_COUNTER
})
```

counterReducer.js

```
import * as types from './actionsTypesConstants'

//NB: the global state of the redux store (store.getState())
//is a sort of map where the keys are "reducerName or reducerKey" and values are
// a "subState" value (number,string,object, array, ...) for a specific reducer
//---> each reducer is associated with its own state (substate of global state)
//---> complementary REDUCERS = complementary SUBJECTS (with their own state)
//---> actions = action VERBS (with attributes/args) to apply on one or more reducer(s).
//NB: when store.dispatch(action) is called , actions are applied on each reducer
//--> action.type like "UPDATE_XYZ" or even "UPDATE_XYZ_REQUEST" ,
//      "UPDATE_XYZ_SUCCESS" , "UPDATE_XYZ_FAILURE"

export function counter(state = 0, action) {
  //console.log(`counter reducer was called with state=${state} and action=${JSON.stringify(action)}`);
  switch (action.type) {
    case types.INCREMENT_COUNTER:
      return state + 1
    case types.DECREMENT_COUNTER:
      return state - 1
    default:
      return state
  }
}
```

Au sein de cet exemple basique :

- le reducer "counter" est une **fonction pure (stateless)** qui admet en entrée l'état existant et une action à appliquer . la valeur de retour correspond au nouvel état construit qui va

remplacer l'ancien .

- l'état est ici est un simple booléen (mais ça peut être une structure objet complexe)
- La valeur par défaut est importante pour initialiser l'état lors d'un premier appel automatique déclenché à l'initialisation du store
- le **default : return state** ; est important de façon à retourner l'état inchangé lorsque cette fonction reducer sera quelquefois appelée avec un type d'action non applicable par ce reducer mais applicable via un autre reducer (dans le cadre d'un assemblage combiné de reducers) .

myCombinedRootReducer.js

```
import { combineReducers } from 'redux';
import { counter } from './counterReducer';
import { confirmOption , myItems} from './itemReducer';

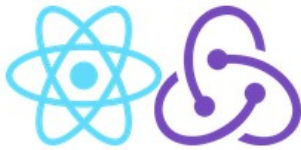
//NB: the global state of the redux store (store.getState())
//is a sort of map where the keys are "reducerName or reducerKey" and values are
// a "subState" value (number,string,object, array, ...) for a specific reducer

export const myRootReducer = combineReducers({
  counter ,
  confirmOption ,
  myItems
})

/*
export const myRootReducer = combineReducers({
  reducerKey1 : reducerPureFunction1,
  reducerPureFunctionName2 : reducerPureFunctionName2,
  counter : counter
})
//simplified as :
export const myRootReducer = combineReducers({
  reducerPureFunctionName2,
  counter
})
*/
```

2. Utilisation de redux au sein de react

npm install --save redux
 npm install --save react-redux



index.js

```
import React from 'react'; import ReactDOM from 'react-dom';
import { createStore } from 'redux' ; import { Provider } from 'react-redux'
import 'bootstrap/dist/css/bootstrap.css'; import './index.css';
import App from './App';
import { myRootReducer } from './myCombinedRootReducer' ;

const store = createStore(myRootReducer);

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

2.1. Principe de fonctionnement "react + redux"

Au sein d'une application "react + redux" bien construite :

- un fichier de type actionsTypesConstants pour bien lister les types d'actions existantes et éviter les doublons
- un rootReducer (voir *myCombinedRootReducer.js* des paragraphes précédents) pour assembler/combinaire un nombre quelquefois conséquent de reducers.
- beaucoup de fichiers xyActions et xyReducer complémentaires .
- pour chaque composant XyComponent visuel de l'application react, une enveloppe complémentaire "*XyContainer*" pour faire le lien avec la partie invisible redux (actions + store + reducers) .

La notion de "XyContainer" est fondamentale dans la liaison "react+redux" .

Un "XyContainer" correspond à une sorte d'enveloppe transparente ajoutant de nouvelles fonctionnalités (on peut le voir comme une application particulière du design pattern "décorateur") . Autrement dit , vis à vis d'un composant parent (exemple "App") , un "XyContainer" s'utilise exactement comme un "XyComponent"/...

Un XyComponent n'est pas du tout relié à redux , il ne fait que gérer l'aspect visuel en utilisant essentiellement (this.props.pa , this.props.pb et this.props.onEvtE1 , this.props.onEvt2) .

Le XyContainer imbrique/enveloppe un XyComponent et permet de paramétrer des associations entre :

- propriétés à afficher et partie du global state (ex : props.pa et state.counter)
- fonctions événementielles à déclencher et application d'une action sur le rootReducer (ex : props.onEvtE1(args) et reducer.dispatch(createE1Action(args))

Grâce à ces paramétrages, les frameworks "react" et "redux" se chargent du reste et vont automatiquement :

- peupler les valeurs des propriétés à afficher avec certaines valeurs actualisées du global state
- déclencher l'application des actions adéquates sur le store de façon à générer un nouvel état bien contrôlé/synchronisé de l'application .

2.2. Exemple de container "redux+react"

CounterComponent.js (composant à envelopper/imbriquer)

```
import React from 'react';
export class CounterComponent extends React.Component{
  render(){
    return ( <div>
      <h3>counter: {this.props.counterValue}</h3>
      <input type="button" value="+" onClick={()=>{this.props.onIncrementCounter()}} /> &nbsp;
      <input type="button" value="-" onClick={()=>{this.props.onDecrementCounter()}} /> &nbsp;
    </div>
    );}}

```

CounterContainer.js

```
import { connect } from 'react-redux' ; //import { bindActionCreators } from 'redux'
import { CounterComponent } from './CounterComponent';
import * as MyActions from './counterActions'

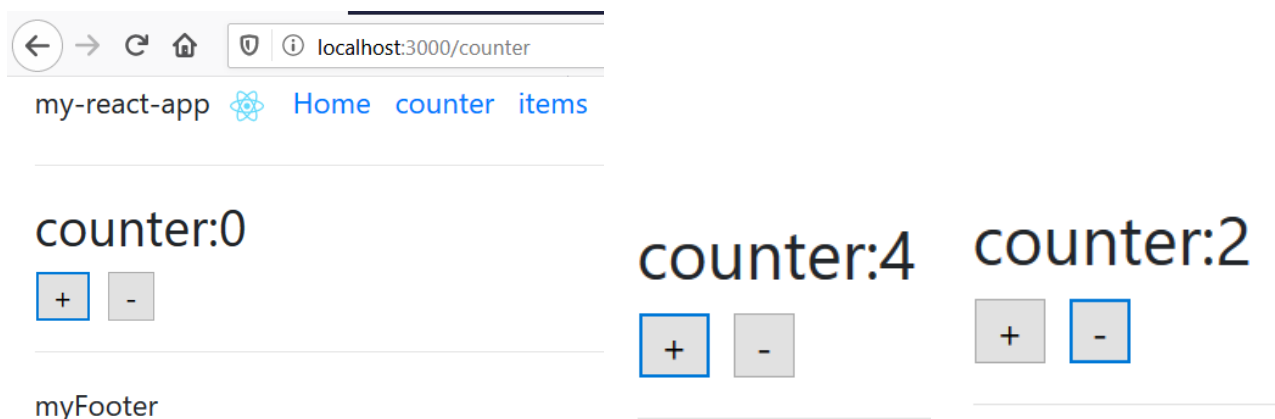
const mapStateToProps = (state)=>{
  return {
    counterValue : state.counter
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    onIncrementCounter: () => { dispatch(MyActions.createIncrementCounterAction()) },
    onDecrementCounter: () => { dispatch(MyActions.createDecrementCounterAction()) }
  }
}

const CounterContainer= connect(mapStateToProps ,mapDispatchToProps)
                           (CounterComponent);
export default CounterContainer ;

```

il suffit ensuite d'utiliser `<CounterContainer />` dans le composant parent *App.js* (ou bien une imbrication indirecte via un `<Router>`) pour obtenir le résultat opérationnel suivant :



2.3. Autre exemple "redux + react"

itemActions.js

```
import * as types from './actionsTypesConstants'

export const createInitItemsAction = () => ({
  type: types.INIT_ITEMS
})

export const createAddItemAction = (obj, cat = 'default') => ({
  type: types.ADD_ITEM,
  obj,
  cat
})

export const createDeleteItemAction = (id) => ({
  type: types.DELETE_ITEM,
  id
})
```

itemReducer.js

```
import * as types from './actionsTypesConstants'

const defaultItemFields = [ 'firstName', 'lastName' ];
const defaultItems = [ { firstName: 'jean', lastName: 'Bon' },
  { firstName: 'alain', lastName: 'Therieur' },
  { firstName: 'alex', lastName: 'Therieur' },
  { firstName: 'axelle', lastName: 'Aire' } ];
```

```

export const myItems = ( state = { itemFields: [], items: [] }, action) => {
  //console.log(`myItems reducer was called with state=${JSON.stringify(state)} and action=${JSON.stringify(action)}`);
  switch (action.type) {
    case types.INIT_ITEMS:
      return { itemFields: defaultItemFields, items: defaultItems };
    case types.DELETE_ITEM:
      return { itemFields: state.itemFields,
        items: state.items.filter((p, i) => { return i !== action.id }) };
    case types.ADD_ITEM:
      let new_items = state.items.slice();
      new_items.push(action.obj);
      const new_state = { itemFields: state.itemFields,
        items: new_items };
      //console.log(`new_state=${JSON.stringify(new_state)}`);
      return new_state;
    default:
      return state;
  }
}

```

ItemComponent.js

```

import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import { MyDataTable } from './my-data-table';

export class ItemComponent extends Component {
  constructor(props){
    super(props);
    this.newItem = { firstName: null, lastName: null };
    props.onInitItems(); //with reducers (dispatch map in decorator ItemContainer)
  }

  reinitNewItem(){
    this.newItem = { firstName: null, lastName: null };
    document.getElementById('firstName').value=null;
    document.getElementById('lastName').value=null;
  }

  render() {
    return (
      <div>
        <div className="row">
          <div className="col-sm-6">
            <h3 className="App">item list</h3>
            <MyDataTable columnNames={this.props.itemFields}
              data={this.props.items}
              onDelete={(index)=>{this.props.onDeleteItems(index);}} />
          </div>

```

```

    <div className="col-sm-6">
      <h3 className="App">new_item</h3>
      firstName: <input id="firstName"
        onChange={(e)=>{this.newItem.firstName=e.target.value;}} /><br/>
      <p></p>
      lastName : <input id="lastName"
        onChange={(e)=>{this.newItem.lastName=e.target.value;}} /><br/>
      <p></p>
      <input type="button" value="add new item"
        onClick={()=>{if(this.newItem.firstName != null &&
          this.newItem.lastName != null) {
            this.props.onAddItem(this.newItem);
            this.reinitNewItem();}
          }} />
    </div>
  </div>
</div>
)
}
}

```

ItemContainer.js

```

import { connect } from 'react-redux'
import { bindActionCreators } from 'redux'
import { ItemComponent } from './ItemComponent';
import * as MyActions from './itemActions'

const mapStateToProps = (state)=>{
  return {
    itemFields : state.myItems.itemFields ,
    items : state.myItems.items
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    onInitItems: () => { dispatch(MyActions.createInitItemsAction()) },
    onDeleteItems: (id) => { dispatch(MyActions.createDeleteItemAction(id)) },
    onAddItem: (item) => { dispatch(MyActions.createAddItemAction(item)) }
  }
}

const ItemContainer = connect(
  mapStateToProps ,
  mapDispatchToProps)(ItemComponent)

export default ItemContainer

```

../..

my-react-app  [Home](#) [counter](#) [items](#)

item list

firstName	lastName	action
jean	Bon	delete
alex	Therieur	delete
axelle	Aire	delete
nom1	prenom1	delete

new_item

firstName:

lastName :

myFooter

ANNEXES

VI - Annexe – rappels javascript / es 2015

1. Mots clefs "let" et "const" (es2015)

Depuis longtemps (en javascript) , le mot clef "**var**" permet de déclarer explicitement une variable dont la portée dépend de l'endroit de sa déclaration (globale ou dans une fonction).

Sans aucune déclaration, une variable (affectée à la volée) est globale et cela risque d'engendrer des effets de bords (incontrôlés) .

Introduits depuis es6/es2015 et typescript 1.4 , les mots clefs **let** et **const** apportent de nouveaux comportements :

- Une variable déclarée via le mot clef **let** a une *portée limité au bloc local* (exemple boucle for) . Il n'y a alors pas de collision avec une éventuelle autre variable de même nom déclarée quelques ligne au dessus du bloc d'instructions (entre {} , de la boucle).
- Une variable déclarée via le mot clef **const** *ne peut plus changer de valeur après la première affectation*. Il s'agit d'une **constante** .

Exemple :

```
const PISur2 = Math.PI / 2;
//PISur2=2; // Error, can't assign to a `const`
console.log("PISur2 = " + PISur2);

var tableau = new Array();
tableau[0] = "abc";
tableau[1] = "def";

var i = 5;
var j = 5;

//for(let i in tableau) {
for(let i=0; i<tableau.length; i++) {
  console.log("*** at index " + i + " value = " + tableau[i] );
}

//for(j=0; j<tableau.length; j++) {
for(var j=0; j<tableau.length; j++) {
  console.log("### at index " + j + " value = " + tableau[j] );
}

console.log("i=" + i); //affiche i=5
console.log("j=" + j); //affiche j=2
```

2. "Arrow function" et "lexical this" (es2015)

Rappels (2 syntaxes "javascript" ordinaires) valables en "javascript/es5" :

```
//Named function:
function add(x, y) {
    return x+y;
}

//Anonymous function:
var myAdd = function(x, y) { return x+y; };
```

Arrow functions (es2015) (alias "Lambda expressions")

Une "**Arrow function**" en javascript/es2015 (à peu près équivalent à une "lambda expression" de java >8) est syntaxiquement introduite via **() => { }**

Il s'agit d'une syntaxe épurée/simplifiée d'une fonction anonyme où les parenthèses englobent d'éventuels paramètres et les accolades englobent le code.

Subtilité du "lexical this" :

La valeur du mot clef "this" est habituellement évaluée lors de l'invocation d'une fonction .

Dans le cas d'une "lambda expression" , le mot clef this est évalué dès la création de la fonction et correspond au this du niveau englobant (classe ou fonction).

Exemples de "lambda expressions" :

```
myFct = (tab) => { var taille = tab.length; return taille; }
//ou plus simplement:
myFct = (tab) => { return tab.length; }
//ou encore plus simplement:
myFct = (tab) => tab.length;
//ou encore plus simplement:
myFct = tab => tab.length;
```

```
var numRes = myFct([12,58,69]);
console.log("numRes=" + numRes); //affiche 3
```

```
myFct2 = (x,y) => { return (x+y) / 2; } //with statement body in { }
//ou plus simplement:
myFct2 = (x,y) => (x+y) / 2; //with simple expression
```

NB : les "**Promise**" (es2015) et la technologie "**RxJs**" utilisée par angular>=2 utilisent beaucoup de "Arrow Functions" .

Exemple de "arrow function" combiné ici avec `arrayXy.forEach(callback)` de "javascript/es5" :

```
let array1 = [ 1 , 2 , 3 , 4 , 5 , 6 ];
let eltPairs = [];
array1.forEach( (e) => { if( (e % 2) === 0 )
                        eltPairs.push(e);
                      }
                );
console.log(eltPairs); // affiche [2,4,6]
```

Suite de l'exemple utilisant `nouveauTableau = arrayXy.map(transform_callback)` de es5 :

```
let eltImpairs = eltPairs.map( v => v-1 );
console.log(eltImpairs); // affiche [1,3,5]
```

Exemple montrant "lexical this" (*arrow function* utilisant *this* de niveau englobant) :

```
var toto = {
  _name: "toto",
  _friends: [ 'titi' , 'tata'],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " est ami avec " + f));
  }
};
toto.printFriends();
```

Autre comportement à connaître:

Si une "fonction fléchée" / "arrow fonction" est à l'intérieur d'une autre fonction, elle partage alors les arguments/paramètres de la fonction parente .

3. for...of (es2015) utilisant itérateurs internes

```
var tableau = new Array();
```

```
//tableau.push("abc");
```

```
//tableau.push("def");
```

```
tableau[0] = "abc";
tableau[1] = "def";
```

Au moins 3 parcours possibles via boucle **for**:

```
var n = tableau.length;
for(let i = 0; i < n; i++) {
  console.log(">> at index " + i + " value = " + tableau[i] );
}
```

```
for(let i in tableau) {
  console.log("** at index " + i + " value = " + tableau[i] );
}
```

//for(index in ...) existait déjà en es5

//for(...of ...) au sens "for each ... of ..." est une nouveauté de es2015

```
for( let s of tableau){
  console.log("## val = " + s );
}
```

NB : la boucle for...of est prédéfinie sur un tableau . il est cependant possible de personnaliser son comportement si l'on souhaite la déclencher sur une structure de données personnalisée. On peut pour cela mettre en oeuvre des itérateurs (et éventuels générateurs de bas niveaux) ---> dans chapitre ou annexe "éléments divers et avancés de es2015" .

3.1. "template string" es2015 (avec quotes inverses et \${})

```
var name = "toto";
var year=2015;
// ES5
//var message = "Hello " + name + " , happy " + year; // Hello toto , happy 2015
// ES6/ES2015 :
const message = `Hello ${name} , happy ${year}`; // Hello toto , happy 2015
//attention: exception "ReferenceError: name is not defined" si name est undefined
console.log(message);
```

\${} peut éventuellement englober des expressions mathématiques ou bien des appels de fonctions.

```
let x=5 , y=6;
let carre = (x) => x*x ;
console.log(`pour x=${x} et y=${y} , x*y=${x*y} et x*x=${carre(x)}`);
//affiche pour x=5 et y=6 , x*y=30 et x*x=25
```

template-string multi-lignes :

```
/*
//ES5
let htmlPart=
"<select> \
  <option>1</option> \
  <option>2</option> \
</select>";
*/
```

```
//template multi-lignes ES2015:
let htmlPart=
`<select>
  <option>1</option>
  <option>2</option>
</select> `;
console.log(htmlPart);
```

3.2. Map , Set

```
// Sets (ensembles sans doublon)
var s = new Set();
s.add("hello").add("goodbye").add("hello");
if(s.size === 2)
  console.log("s comporte 2 elements");
if(s.has("hello"))
  console.log("s comporte hello");
```

// List ==> Array ordinaire (déjà en es5 , à remplir via .push()).

```

// Maps (table d'association (clef,valeur))
var m = new Map();
m.set("hiver", "froid , neige");
m.set("printemps", "fleur , vert");
m.set("ete", "soleil , plage");
m.set("ete", "chaud , plage"); //la nouvelle valeur remplace l'ancienne .
m.set("automne", "feuilles mortes");
let carateristique_ete = m.get("ete");
console.log("carateristique_ete="+carateristique_ete); //chaud , plage
if(m.has("ete"))
    console.log("Map m comporte une valeur associée à ete");
for(saison of m.keys()){
    console.log("saison "+ saison + " - " + m.get(saison));
}
//m.values() permettrait d'effectuer une boucle sur les valeurs (peu importe les clefs)
for([k,v] of m.entries()){
    console.log("saison "+ k + " -- " + v);
}
m.forEach((val,key)=> console.log("saison "+ key + " --- " + val));
m.clear();
if(m.size===0)
    console.log("map m is empty");

//Bien que ce code soit lisible et explicite, un vieil objet javascript en faisait autant :
var objectMap = {
    hiver : "froid , neige",
    printemps : "fleur , vert",
};
objectMap["ete"]="chaud, plage" ;
console.log("carateristique_hiver="+ objectMap["hiver"]); // froid , neige

//Une des valeurs ajoutées par "Map" (es2015) est la possibilité d'avoir des clefs de n'importe
//quelle sorte possible (ex : window , document , element_arbre_DOM, ...).

```

NB : es2015 a également introduit les variantes "WeakMap" et "WeakSet" mais celles-ci ne sont utilisables et utiles que dans des cas très pointus (ex : programmation de "cache").

"WeakMap" et "WeakSet" sont exposés dans le chapitre (ou annexe) "Aspects divers et avancés" .

3.3. "Destructuring" (affectation multiple avec perte de structure)

Destructuring objet : extract object parts in several variables :

```
const p = { nom : 'Allemagne' , capitale : 'Berlin' , population : 83000000, superficie : 357386};
const { nom , capitale } = p;
console.log("nom="+nom+" capitale="+capitale);
//nom="?" ; interdit car nom et capitale sont considérées comme des variables "const"

//NB: les noms "population" et "superficie" doivent correspondre à des propriétés de l'objet
//dont il faut (partiellement) extraire certaines valeurs (sinon "undefined")
//l'ordre n'est pas important
const { superficie , population } = p;
console.log("population="+population+" superficie="+superficie);
==>
```

nom=Allemagne capitale=Berlin
population=83000001 superficie=357386

utilité concrète (parmi d'autres) : *fonction avec paramètres nommés* :

```
function fxabc_with_named_param( { paramX=0 , a=0 , b=0 , c=0 } = {} ){
    //return ax^2+bx+c
    return a * Math.pow(paramX,2) + b * paramX + c;
}

let troisFois4 = fxabc_with_named_param( { paramX :4 , b : 3 } );
console.log("troisFois4="+troisFois4 );//12
let deuxFois4AuCarreplus6 = fxabc_with_named_param( { paramX :4 , a : 2 , c :6 } );
console.log("deuxFois4AuCarreplus6="+deuxFois4AuCarreplus6 );//38
```

Destructuring iterable (array or ...) :

```
const [ id , label ] = [ 123 , "abc" ];
console.log("id="+id+" label="+label);

//const arrayIterable = [ 123 , "abc" ];
//var iterable1 = arrayIterable;
const stringIterable = "XYZ";
var iterable1 = stringIterable;
const [ partie1 , partie2 ] = iterable1;
console.log("partie1="+partie1+" partie2="+partie2);
```

==>

id=123 label=abc
partie1=X partie2=Y



Autre exemple plus artistique (Picasso) :

3.4. for (..of ..) with destructuring on Array , Map, ...

```
const dayArray = ['lundi', 'mardi', 'mercredi'];
for (const entry of dayArray.entries()) {
  console.log(entry);
}
// [ 0, 'lundi' ]
// [ 1, 'mardi' ]
// [ 2, 'mercredi' ]

for (const [index, element] of dayArray.entries()) {
  console.log(` ${index}. ${element} `);
}
// 0. lundi
// 1. mardi
// 2. mardi
```

```
const mapBoolNoYes = new Map([
  [false, 'no'],
  [true, 'yes'],
]);
for (const [key, value] of mapBoolNoYes) {
  console.log(` ${key} => ${value} `);
}
// false => no
// true => yes
```

4. Prog. orientée objet "es2015" (class, extends, ...)

4.1. Préambule ("poo via prototype es5" --> "poo es2015")

La programmation orientée objet existait déjà en "javascript/es5" mais avec une syntaxe très complexe , verbeuse et peu lisible (prototypes avancés).

Dès l'époque "es5" , le mot clef constructor existait et l'on pouvait même définir une relation d'héritage avec une syntaxe complexe et rebutante.

La nouvelle version "es6/es2015" du langage "javascript/ecmascript" a enfin apporté une nouvelle syntaxe "orientée objet" beaucoup plus claire et lisible donnant envie de programmer de nouvelles classes d'objet en javascript moderne.

4.2. Classe et instances

```
class Compte{
  constructor(numero,label,solde){
    this.numero = numero;
    this.label=label;
    this.solde=solde;
  }

  debiter(montant) {
    this.solde -= montant; // this.solde = this.solde - montant;
  }

  crediter(montant) {
    this.solde += montant; // this.solde = this.solde + montant;
  }
}
```

```
let c1 = new Compte(); //instance (exemplaire) 1
console.log("numero et label de c1: " + c1.numero + " " + c1.label); // undefined undefined
console.log("solde de c1: " + c1.solde); // undefined

let c2 = new Compte(); //instance (exemplaire) 2
c2.solde = 100.0;
c2.crediter(50.0);
console.log("solde de c2: " + c2.solde); //150.0

let c3 = new Compte(3,"compte3",300); //instance (exemplaire) 3
console.log("c3: " + JSON.stringify(c3)); //{"numero":3,"label":"compte3","solde":300}
```

NB: Sans initialisation explicite (via constructeur ou autre) , les propriétés internes d'un objet sont par défaut à la valeur "undefined" .

4.3. "constructor" avec éventuelles valeurs par défaut

Un constructeur est une méthode qui sert à initialiser les valeurs internes d'une instance dès sa construction (dès l'appel à new) .

En langage javascript/es2015 le constructeur se programme comme la méthode spéciale "constructor" (mot clef du langage) :

```
class Compte{
  constructor(numero,label,solde){
    this.numero = numero; this.label=label; this.solde=solde;
  }
}
```

```
//...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
c1.crediter(50.0); console.log("solde de c1: " + c1.solde);
```

NB: contrairement au langage java (où la surcharge y est permise) , il n'est pas possible d'écrire plusieurs versions du constructeur (ou d'une fonction de même nom) en javascript/es2015:

```
constructor(numero, libelle, soldeInitial){
    this.numero = numero;
    this.label = libelle;
    this.solde = soldeInitial;
}
```

```
constructor(){
    this.=0;
    this.label="?";
    this.=0.0;
}
```

Il faut donc quasi systématiquement utiliser la syntaxe = *valeur_par_defaut* sur les arguments d'un constructeur pour pouvoir créer une nouvelle instance en précisant plus ou moins d'informations lors de la construction :

```
class Compte{
    constructor(numero=0, libelle="?", soldeInitial=0.0){
        this.numero = numero;
        this.label = libelle;
        this.solde = soldeInitial;
    }//...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
var c2 = new Compte(2,"compte 2");
var c3 = new Compte(3); var c4 = new Compte();
```

4.4. propriétés (pseudo attributs - mots clefs "get" et "set")

Bien que "es2015" ne prenne pas en charge les mots clefs "~~public~~", "~~private~~" et "~~protected~~" au niveau des membres d'une classe et que toutes les méthodes définies soient publiques , il est néanmoins possible d'utiliser les mots clefs **get** et **set** de façon à ce qu'un **couple de méthodes "get xy()" et "set xy(...)"** soit vu de l'extérieur comme une propriété (pseudo-attribut "xy") de la classe .

Attention : contrairement au langage java , il ne s'agit pas de convention de nom getXy() / setXy() mais de véritables mot clefs "get" et "set" à utiliser comme préfixe (avec un espace) .

```
class Compte{
```



```

    get decouvertAutorise(){
        return (this._decouvertAutorise!==undefined)?this._decouvertAutorise:0;
    }
    set decouvertAutorise(decouvertAutorise){
        this._decouvertAutorise = decouvertAutorise;
    }
//... }

```

```

let c4=new Compte() ;
c4.decouvertAutorise = -300;
let decouvertAutorisePourC4 = c4.decouvertAutorise;
console.log("decouvertAutorisePourC4="+decouvertAutorisePourC4);

```

NB :

- il est possible de ne coder que le "get xy()" pour une propriété en lecture seule .
- il faut un nom différent (avec par exemple un "_" en plus) au niveau du nom de l'attribut interne préfixé par "this." car sinon il y a confusion entre attribut et propriété et cela mène à des boucles infinies .
- un "getter" peut éventuellement être supprimé via le mot clef **delete** (ex : delete obj.dernier)
- **Object.defineProperty()** permet (dans le cas pointus) de définir un "getter" par "méta-programmation" (ex : dans le cadre d'un framework ou d'une api générique).

En javascript es2015, le mot clef **get** sert surtout à définir **une propriété dont la valeur est calculée dynamiquement** :

```

var obj = {
    get dernier() {
        if (this.arrayXy.length > 0) {
            return this.arrayXy[this.arrayXy.length - 1];
        }
        else {
            return null;
        }
    },
    arrayXy: ["un","deux","trois"]
}

console.log("dernier="+obj.dernier); // "trois"

```

4.5. mot clef "static" pour méthodes de classe

De la même façon que dans beaucoup d'autres langages orientés objets (c++, java, ...) , le mot clef **static** permet de définir des méthodes de classe (dont l'appel s'effectue avec le préfixe "NomDeClasse." plutôt que "instancePrecise.").

ES2015 ne permet pas d'utiliser static avec un attribut mais on peut utiliser "static" sur une propriété (avec mot clef get et éventuellement set). Dans ce cas la valeur de la propriété sera partagée par toutes les instances de la classe (et l'accès se fera via le préfixe "NomDeClasse.")

Exemple:

```
class CompteEpargne {
    //...
    static get tauxInteret(){
        return CompteEpargne.prototype._tauxInteret;
    }
    static set tauxInteret(tauxInteret){
        CompteEpargne.prototype._tauxInteret=tauxInteret;
    }

    static get plafond(){
        return CompteEpargne.prototype._plafond;
    }
    static set plafond(plafond){
        CompteEpargne.prototype._plafond=plafond;
    }

    static methodeStatiqueUtilitaire(message){
        console.log(">>>" + message + "<<<");
    }
}

CompteEpargne.prototype._tauxInteret = 1.5 ; //1.5% par default
CompteEpargne.prototype._plafond = 12000; //par default
```

```
let cEpargne897 = new CompteEpargne() ;
//cEpargne897.solde = 250.0; // instancePrecise.proprieteOrdinairePasStatique
console.log("taux interet courant=" + CompteEpargne.tauxInteret); //1.5
console.log("plafond initial=" + CompteEpargne.plafond); //12000
CompteEpargne.plafond = 10000;
let messagePlafond= "nouveau plafond=" + CompteEpargne.plafond; //10000
CompteEpargne.methodeStatiqueUtilitaire(messagePlafond);
```

NB :

- En cas d'héritage , une sous classe peut faire référence à une méthode statique de la classe parente via le mot préfixe `super`.
- Une méthode statique ne peut pas être invoquée avec le préfixe `this` .

4.6. héritage et valeurs par défaut pour arguments:

```
class Animal {
  constructor(theName="default animal name") {
    this.name= theName;
  }
  move(meters = 0) {
    console.log(this.name + " moved " + meters + "m.");
  }
}
```

```
class Snake extends Animal {
  constructor(name) { super(name); }
  move(meters = 5) {
    console.log("Slithering...");
    super.move(meters);
  }
}
```

```
class Horse extends Animal {
  constructor(name) { super(name); }
  move(meters = 45) {
    console.log("Galloping...");
    super.move(meters);
  }
}
```

```
var a = new Animal(); //var a = new Animal("animal");

var sam = new Snake("Sammy the Python"); //var sam = new Snake();

var tom = new Horse("Tommy the Palomino");

a.move() ; // default animal name moved 0m.
sam.move(); // Slithering... Sammy the Python moved 5m.

tom.move(34); //avec polymorphisme (for Horse)
// Galloping... Tommy the Palomino moved 34m.
```

4.7. Object.assign()

Object.assign(obj,otherObject) ; permet (selon les cas) de :

- effectuer un clonage en mode "shallow copy" (copies des références vers propriétés)
- ajouter dynamiquement un complément
- ajouter le comportement d'une classe à un pur objet de données

Exemple d'objet original:

```
const subObj = { pa : "a1" , pb : "b1" };
const obj1 = { p1: 123 , p2 : 456 , p3 : "abc" , subObj : subObj };
```

Clonage imparfait (pas toujours en profondeur) avec Object.assign(clone,original)

```
var objCloneViaShallowCopy = {}
Object.assign(objCloneViaShallowCopy,obj1); //copy of property reference
console.log("clonage via assign / shallowCopy=" + JSON.stringify(objCloneViaShallowCopy));
// {"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a1","pb":"b1"}}
objCloneViaShallowCopy.subObj.pa="a2";
//modification à la fois sur objCloneViaShallowCopy.subObj et sur obj1.subObj
console.log("obj1" + JSON.stringify(obj1));
// {"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a2","pb":"b1"}}
objCloneViaShallowCopy.subObj.pa="a1"; //restituer ancienne valeur
```

Clonage en profondeur avec obj=JSON.parse(Json.stringify(original)) :

```
var obj = JSON.parse(JSON.stringify(obj1)); //clonage en profondeur
console.log("clonage en profondeur=" + JSON.stringify(obj));
obj.subObj.pa="a2"; //modification que sur obj.subObj
console.log("obj1" + JSON.stringify(obj1)); //obj1 inchangé ( "a1")
```

Ajout de données via Object.assign(obj, complément) :

```
Object.assign(obj, { p4: true , p5: "def" });
console.log("après assign complement=" + JSON.stringify(obj));
// {"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a2","pb":"b1"},"p4":true,"p5":"def"}
```

Ajout comportemental via obj=Object.assign(new MyClass(), obj):

```
class Pp {
    constructor(p1=0,p2=0,p3=0){
        this.p1=p1; this.p2=p2; this.p3=p3;
    }

    sumOfP1P2P3(){
        return this.p1+this.p2+this.p3;
    }
}

const subObj = { pa : "a1" , pb : "b1" };
const obj1 = { p1: 123 , p2 : 456 , p3 : "abc" , subObj : subObj };
obj = JSON.parse(JSON.stringify(obj1));//réinitialisation du clone "obj"
if(!(obj instanceof Pp))
    console.log("obj is not instance of Pp , no sumOfP1P2P3() method");
//console.log("obj.sumOfP1P2P3()="+obj.sumOfP1P2P3()); not working
obj = Object.assign(new Pp(),obj);
if((obj instanceof Pp)) console.log("obj is instance of Pp , with sumOfP1P2P3() method");
console.log("obj.sumOfP1P2P3()="+obj.sumOfP1P2P3()); //ok : 579abc
```

Mixin set of additional methods with Object.assign(C1.prototype , mixinXyz):

exemple :

```
class C1 {
    constructor(id=null,label="?"){
        this.id=id; this.label=label;
    }

    displayId(){
        console.log(`id=${this.id}`);
    }
}
```

```

let myMixin = {
  //mixin object = set of additional methods (without real inheritance):

  labelToUpperCase(){
    this.label = this.label.toUpperCase();
  },
  displayLabel(){
    console.log(`label=${this.label}`);
  }
}

let objet = new C1(1,"abc");
//objet.displayLabel();//not a method of C1
Object.assign(C1.prototype,myMixin); //ajouter méthodes de myMixin à C1
objet.displayId();
objet.labelToUpperCase();
objet.displayLabel(); //ABC
let objetBis = new C1(2,"def");
objetBis.displayId();
objetBis.displayLabel();//def

```

4.8. Notions "orientée objet" qui ne sont pas gérées pas es2015

Les éléments "orientés objets" suivants ne sont pas pris en charge par un moteur javascript/es2015 mais sont pris en charge par le langage "typescript" (".ts" à traduire en ".js" via babel ou "tsc") :

- classes et méthodes abstraites (mot clef "abstract")
- visibilités "public" , "private" , "protected"
- interfaces (mots clefs "interface" et "implements")
- "public" , "private" ou "protected" au niveau des paramètres d'un constructeur pour définir automatiquement certaines variables d'instances (attributs)
- ...

5. Modules (es2015)

5.1. Types de modules (cjs , amd , es2015 , umd , ...)

Beaucoup de technologies javascript modernes s'exécutent dans un environnement prenant en charge des modules (bien délimités) de code (avec import/export) . Le développement d'une application "Angular2+" s'effectue à fond dans ce contexte.

Les principales technologies de "modules javascript" sont les suivantes :

- **CommonJS (cjs)** – modules "synchrones" , **syntaxe** "var xyz = **requires**('xyz')"
NB : node (nodeJs) utilise partiellement les idées et syntaxes de CommonJS .
- **AMD** (Asynchronous **M**odule **D**efinition) avec chargements asynchrones
- **ES2015 Modules** : syntaxiquement standardisé , mots clef "import {...} from '...'" et export pour la gestion dynamique des modules(possibilité de générer des bundles (es5 ou es6) regroupant plusieurs modules statiquement assemblés ensemble).
- **SystemJS** (très récent et pas encore complètement stabilisé) supporte en théorie les 3 technologies de modules précédentes (cjs , amd, es2015) . SystemJS nécessite certains paramétrages (quelquefois complexes) et s'utilise assez souvent avec gulp .
--> Attention : SystemJS s'est révélé assez instable et n'est pas toujours ce qui y a de mieux .

Il existe aussi les **formats de modules suivants** :

- **umd** (universal **m**odule **d**efinition) – *fichiers xyz.umd.js*
- **iife** (immediately-invoked **f**unction **e**xpression) – *fonctions anonymes auto-exécutées*

5.2. Modules "es6/es2015" et organisation en fichiers

Les modules ES6 :

- ont une syntaxe simple et sont basés sur le découpage en fichiers (un module = un fichier),
- sont automatiquement en mode « strict » (rigoureux),
- offrent un support pour un chargement asynchrone et permet de générer des bundles "statiques" via rollup ou webpack .

Les modules doivent exposer leurs variables et méthodes de façon explicite. On dispose donc des deux mots clés :

- **export** : pour exporter tout ce qui doit être accessible en dehors du module,
- **import** : pour importer tout ce qui doit être utilisé dans le module (et qui est donc exporté par un autre module).

5.3. Exemple avec chargement direct depuis navigateur récent:

math-util.js

```
export function additionner(x , y) {  
  return x + y;  
}
```

```
export function multiplier(x, y) {
  return x * y;
}
```

ou bien

```
function additionner(x, y) {
  return x + y;
}
function mult(x, y) {
  return x * y;
}
export { additionner, mult as multiplier };
```

dom-util.js

```
export class DomUtil {
  static displayInDiv(divId,message){
    document.querySelector('#'+divId).innerHTML = message;
  }

  static multilineMessage(...args){
    //NB: la syntaxe ... permet de récupérer tous (ou bien les derniers) arguments (en nombre variable)
    //sous forme de tableau . Cette syntaxe est permise en mode "strict" alors que la
    //syntaxe Dom.multilineMessage.arguments est interdite en mode "strict" (dans module es6)
    let nb_arg=args.length;
    let messages=null;
    if(nb_arg>=1) messages=args[0];
    for(let i=1;i<nb_arg;i++)
      messages+="<br/>" + args[i];
    return messages;
  }
}
```

main.js

```
import { additionner as add, multiplier } from "./math-util.js";
import { DomUtil } from "./dom-util.js";

function carre(x){
  return multiplier(x,x) ;
}
```



```

}
/*
var msg1 = "Le carre de 5 est " + carre(5); console.log(msg1);
var msg2 = "4 * 3 vaut " + multiplier(4, 3); console.log(msg2);
var msg3 = "5 + 6 vaut " + add(5, 6); console.log(msg3);
document.querySelector('#divA').innerHTML = msg1 + "<br/>" + msg2 + "<br/>" + msg3;
*/
DomUtil.displayInDiv('divA',
    DomUtil.multilineMessage(
        "Le carre de 5 est " + carre(5),
        "4 * 3 vaut " + multiplier(4, 3),
        "5 + 6 vaut " + add(5, 6)
    ));

```

```

<html>
  <body>
    <script type="module" src="main.js"></script>
    <div id="divA"></div>
  </body>
</html>

```

Attention :

- `type="module"` est **indispensable** mais **n'est supporté que par les navigateurs récents**
- la page html et les modules javascripts doivent être téléchargés via http (par exemple via `http://localhost:3000/` et `lite-server`) .

5.4. default export (one per module)

xy.js

```
export function mult(x, y) {
  return x * y;
}

//export default function_or_object_or_class (ONE PER MODULE)
export default {
  name : "xy",
  features : { x : 1 , y: 3 }
}
```

main.js

```
import xy , { mult } from "./xy.js";
...
let msg = xy.name + "--" + JSON.stringify(xy.features) + "--" + mult(3,4);
```

5.5. Agrégation de modules

mod1.js

```
export function f1(msg) { return "*" + msg }
export function f2(msg) { return "*" + msg; }
export function f2bis(msg) { return "*" + msg; }
```

mod2.js

```
export function f3(msg) { return "#" + msg }
export function f4(msg) { return "##" + msg; }
```

mod1-2.js

```
//agrégation de modules : mod1-2 = mod1 + mod2
//importer certains éléments du module "mod2" et les ré-exporter:
export { f1, f2 } from "./mod1.js"
//importer tous les éléments du module "mod2" et les ré-exporter tous :
export * from "./mod2.js"
```

main.js

```
import { f1 , f2 , f3 , f4 } from "./mod1-2.js";
let f_msg=f1('abc')+'-'+f2('abc')+'-'+f3('abc')+'-'+f4('abc');
```

ou bien

```
import * as f from "./mod1-2.js";
let f_msg=f.f1('abc')+'-'+f.f2('abc')+'-'+f.f3('abc')+'-'+f.f4('abc');
```

5.6. Technologies de "packaging" (webpack , rollup, ...) et autres

De façon à éviter le téléchargement d'une multitude de petits fichiers , il est possible de créer des gros paquets appelés "**bundles**" .

Les principales technologies de packaging "javascript" sont les suivantes :

- **webpack** (mature et supportant les modules "csj" , "amd" , ...)
- **rollup** (récent et pour modules "es2015") . Rollup est une fusion intelligente de n fichiers en 1 (remplacement des imports/exports par sous contenu ajustés , prise en compte de la chaîne des dépendances en partant par exemple de main.js)
- **SystemJs-builder** (technologie assez récente et un peu moins mature)

Autres technologies annexes (proches) :

- **browserify** : technologie déjà assez ancienne permettant de faire fonctionner un module "nodeJs" dans un navigateur après transformation.
- **babel** : transformation (par exemple es2015 vers es5)
- **uglify** : minification (enlever tous les espaces et commentaires inutiles, simplifier noms des variables, ...) → code beaucoup plus compact (xyz.min.js) .
- **gzip** : compression .Les fichiers bundlexy.min.js.gz sont automatiquement traités par quasiment tous les serveurs HTTP et les navigateurs : décompression automatique après transfert réseau).

5.7. Packaging web via rollup / es2015 et npm

L'un des principaux atouts de la structure des "modules es2015" tient dans les imports statiques et précis qui peuvent ainsi être analysés pour une génération optimisée des bundles à déployer en production.

Au lieu d'écrire `var/const xyModule = requires('xyModule')` comme en "cjs", la syntaxe plus précise de es2015 permet d'écrire

import { **Composant1** , ... , **ComposantN** } **from** 'xyModule' .

Ainsi l'optimisation dite "**Tree-Shaking**" permet d'exclure tous les composants jamais utilisés de certaines librairies et la taille des bundles générés est plus petite.

La technologie de packaging "rollup" qui est spécialisée "es2015" peut ainsi exploiter cette optimisation via la chaîne de transformation suivante :

```
main[.es2015].js
  with import           → rollup → myBundle.es2015.js → myBundle.es5.js
subModuleXx[.es2015].js
subModuleYy[.es2015].js      (*)
```

(*) **es2015-to-es5** via **babel** (*presets* : *es2015* ou *[env]*) ou autres permet d'obtenir un bundle interprétable par quasiment tous les navigateurs.

```
npm install -g rollup
npm init
npm install --save-dev babel-cli
npm install --save-dev babel-preset-env
```

package.json

```
{
  "name": "with-modules-and-rollup",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "es6bundle-to-es5": "babel dist/build-es2015 -d dist/build-es5",
    "build" : "rollup --config rollup.config.js && babel dist/build-es2015 -d dist/build-es5"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.7.0"
  }
}
```

```
rollup --config rollup.config.js
```

ou bien

```
npm run build
```

rollup.config.js

```
export default {
  input: 'src/main.js',
  output : {
    file: 'dist/build-es2015/main-bundle.js',
    format: 'iife'
  }
};
```

.babelrc

```
{
  "presets": ["env"]
}
```

Dans la configuration *rollup.config.js* ,

input :.../main.js correspond au point d'entrée (.js) autrement dit la racine d'un arbre import/export entre différents fichiers (es6) qui seront analysés et gérés par rollup.

Il peut quelquefois y avoir plusieurs input/output dans le fichier rollup.config.js.

le **format** peut être "cjs" pour une future interprétation via node/nodeJs

ou "iife" pour une future interprétation via html/js (navigateur)

5.8. Packaging web via webpack

--> voir chapitre ou annexe suivante ...

5.9. Pattern module et IIFE (compatible es5)

IIFE (i.e. *Immediately-Invoked Function Expression* ou Expression de fonction invoquée immédiatement) permet de mettre en oeuvre le pattern "module" dans une syntaxe comprise par n'importe quel navigateur (de niveau es5) :

Une **IIFE** s'écrit de cette façon:

```
var ModuleXy = (function() {
  // Bloc de code à exécuter
})();
```

Pour utiliser une IIFE pour définir un module, on peut écrire par exemple:

```
var ModuleXy = (function() {
  var self = {};
  function privateFunction() {
    // ...
  };

  self.publicFunc = function() {
    privateFunction();
  };
});
```

```
    return self;
  })();
```

Cette écriture permet de définir une variable appelée `ModuleXy` qui va contenir des membres et des fonctions, ce qui correspond à la notion de module:

- `privateFunction()` est une fonction privée.
- `publicFunc()` est une fonction publique.

Pour utiliser ce type de module, on peut écrire

```
moduleXy.publicFunc();
```

6. Promise (es2015)

6.1. L'enfer des "callback" (sans promesses) :

Beaucoup d'api javascript ont été conçues pour fonctionner en mode asynchrone (sans blocage). Par exemple, pour séquentiellement saisir `x`, saisir `y` et calculer `x+y`, le code nécessaire qui serait très simple et très lisible en C/C++ ou java est assez complexe dans l'environnement node-js :

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask(question, callback) {
    stdin.resume();
    stdout.write(question + ": ");
    stdin.once('data', function(data) {
        data = data.toString().trim();
        callback(data);
    });
}

//utilisation chaînée avec callbacks imbriquées:
ask("x", function(valX){
    var x=Number(valX);
    ask("y", function(valY){
        var y=Number(valY);
        var res=x+y ;
        console.log("res = (x+y)="+res);
        process.exit();
    });
});
```

```
});
});
```

6.2. Principe de fonctionnement des promesses (*Promise*)

Lorsque l'on déclenche via un appel de fonction un traitement asynchrone dont le résultat ne sera prêt/connu que dans le futur, on peut retourner immédiatement un objet de type "Promise" qui encapsule l'attente d'une réponse promise.

Le résultat promis qui sera récupéré en différé dans le temps est soit une réponse positive (promesse tenue) soit une réponse négative (erreur / promesse rompue).

A l'intérieur de la fonction asynchrone appelée, on crée et retourne une promise via l'instruction

```
return new Promise((resolve,reject) => { if(...) resolve(...) else reject(...) ; } ) ;
```

//où resolve et reject sont des noms logiques de callbacks appelées dans le futur

//pour transmettre l'issue positive ou négative du traitement asynchrone.

A l'extérieur, l'appel s'effectue via la syntaxe

```
.then((resolvedValue)=>{ ....} , (rejectedValue) => { ... } ) ;
```

ou bien

```
.then((resolvedValue)=>{ ....})
```

```
.catch((rejectedValue) => { ... } ) ;
```

NB: Si à l'intérieur d'un `.then(()=>{...})` on appelle et retourne une fonction asynchrone retournant à son tour une autre "Promise", on peut alors enchaîner d'une manière lisible une séquence d'appel à d'autres `.then()` qui seront alors exécutés les uns après les autres au fur et à mesure de la résolution des promesses asynchrones :

```
appelAsynchrone1RetournantPromesse1(...)
```

```
.then((resPromesse1)=>{ .... ; return appelAsynchrone2RetournantPromesse2(....);})
```

```
.then((resPromesse2)=>{ .... ; return appelAsynchrone2RetournantPromesse3(....);})
```

```
.then((resPromesse3)=>{ .... ; })
```

```
.catch((premiereErreurPromesse1ou2ou3)=>{...});
```

NB: avant d'exister en version normalisée "es2015", les "Promises" avaient été prises en charge via l'ancienne bibliothèque "q" (`var deferred = Q.defer(); deferred.resolve(data); ... return deferred.promise;`) avec même utilisation `.then(...).then(...).catch(...)`.

Les "Promises" étaient déjà beaucoup utilisées à l'époque de es5/angular-js (avant 2015 et es6/es2015).

Exemple (avec Promise/es2015) :

```
var stdin = process.stdin;
var stdout = process.stdout;
```

```
function ask_(question) {
```

```
    return new Promise ((resolve,reject)=> {
```

```
        stdin.resume();
```

```
        stdout.write(question + ": ");
```

```
        stdin.once('data', function(data) {
```

```
            data = data.toString().trim();
```

```
            if(data=="fin")
```

```

        reject("end/reject");
      } else {
        resolve(data);
      }
    });
  });
}

var x,y,z;
//calcul (x+y)*z après enchaînement lisible (proche séquentiel) de "saisir x", "saisir y", "saisir z":
ask_("x")
.then((valX)=>{ x=Number(valX); return ask_("y");})
.then((valY)=> { y=Number(valY); let res=x+y ;
                console.log("(x+y)=" +res);
                return ask_("z");
              })
.then((valZ)=> { z=Number(valZ); let res=(x+y)*z ;
                console.log("(x+y)*z=" +res);
                process.exit();
              })
.catch((err)=>{console.log(err);process.exit();});

```

6.3. Autre exemple simple (sans et avec "Promise"):

```

function strDateTime() {
  return (new Date()).toLocaleString();
}

const affDiffere = () => {
  setTimeout (()=> {console.log("after 2000 ms " + strDateTime());} , 2000);
};

```

Version sans "Promise" avec callbacks imbriquées :

//NB : via .setTimeout(...., delay) et return { responseJsData } ; on simule ici une récupération de //données via un appel asynchrone vers une base de données ou un WS REST.

```

const getUserInCb =
  (cbWithName) => {
    setTimeout (()=> { cbWithName({ name : "toto" });} , 2000);
  };

const getAddressFromNameInCb =
  (name , cbWithAddress) => {
    setTimeout (()=> { cbWithAddress({ adr : "75000 Paris for name="+name });}
      , 1500);
  };

console.log("debut :" + strDateTime() );

```



```

affDiffere();

getUserInCb(
  (user) => {
    console.log("username=" + user.name);
    getAddressFromNameInCb(user.name,
      (address) => { console.log("address=" + address.adr ); }
    );
  }
);

console.log("suite :" + strDateTime() );

```

résultats:

```

debut :2019-4-23 16:46:24
suite :2019-4-23 16:46:24
after 2000 ms 2019-4-23 16:46:26
username=toto
address=75000 Paris for name=toto

```

Même exemple avec "Promise es6/es2015" :

```

function getUserFromIdAsPromise(id){
  return new Promise (
    (resolveCbWithName,rejectCb) => {
      setTimeout (()=> { if(id) resolveCbWithName({ name : "toto" });
                        else rejectCb("id should not be null!");
                        }, 2000);
    });
}

function getAddressFromNameAsPromise(name){
  return new Promise (
    (resolveCbWithAddress) => {
      setTimeout (()=> { resolveCbWithAddress({ adr : "75000 Paris for name="
                                                +name });}, 1500);
    });
}

console.log("debut :" + strDateTime() ); affDiffere();

getUserFromIdAsPromise(1)
//getUserFromIdAsPromise(null)
  .then( (user) => { console.log("username=" + user.name);
                    //returning new Promise for next then() :
                    return getAddressFromNameAsPromise(user.name);
                  })
  .then( (address) => { console.log("address=" + address.adr ); } )

```

```
.catch(error => { console.log("error:" + error); });
console.log("suite :" + strDateTime() );
```

Résultats avec id=1 :

```
debut :2019-4-23 17:18:44
suite :2019-4-23 17:18:44
after 2000 ms 2019-4-23 17:18:46
username=toto
address=75000 Paris for name=toto
```

Résultats avec id=null :

```
debut :2019-4-23 17:33:17
suite :2019-4-23 17:33:17
after 2000 ms 2019-4-23 17:33:19
error:id should not be null!
```

L'exemple ci-dessus montre que :

- l'on peut nommer comme on le souhaite les callbacks "resolve" et "reject" . Ce qui permet quelquefois de rendre le code plus intelligible
- la callback "reject" est facultative

6.4. Propriétés des "Promises"

```
fairePremiereChose()
.then(result1 => faireSecondeChose(result1))
.then(result2 => faireTroisiemeChose(result2))
.then(finalResult3 => {
  console.log('Résultat final : ' + finalResult3);
})
.catch(failureCallback);
```

où

```
(resultatAppelPrecedent) => faireNouvelleChose(resultatAppelPrecedent)
```

est synonyme de

```
(resultatAppelPrecedent) => { return faireNouvelleChose(resultatAppelPrecedent) ; }.
```

Si aucun catch , il est éventuellement possible de traiter l'événement "**unhandledrejection**"

```
window_or_worker.addEventListener("unhandledrejection", event => {
  // Examiner la ou les promesse(s) qui posent problème en debug
  // Nettoyer ce qui doit l'être quand ça se produit en réel
}, false);
```

Dans des cas "ultra simples" ou "triviaux" , on pourra éventuellement créer et retourner ***une promesse à résolution ou rejet immédiat*** avec une syntaxe de ce type :

```
argValue => Promise.resolve(argValue);
```

```
//version abrégée de argValue => new Promise((resolve)=>resolve(argValue))
```

```
errMsg => Promise.reject(errMsg);
```

6.5. Compositions (all , race, ...)

On peut déclencher des traitements asynchrones en parallèle et attendre que tout soit fini pour analyser globalement les résultats :

```
Promise.all([func1(), func2(), func3()])
  .then(([resultat1, resultat2, resultat3]) => { /* utilisation de resultat1/2/3 */ });
```

La variante ci-après permet de déclencher des traitements asynchrones en parallèle et attendre que le premier résultat (retourné par la fonction asynchrone la plus rapide) :

```
Promise.race([func1(), func2()])
  .then( (firstReturnedValue) => { console.log(firstReturnedValue); });
```

Exemple :

```
function getUppercaseDataAfterDelay(data, delay){
  return new Promise (
    (resolve) => {
      setTimeout (()=> { resolve(data.toUpperCase());}, delay);
    });
}

Promise.all( [ getUppercaseDataAfterDelay("abc",2000) ,
  getUppercaseDataAfterDelay("def",1500) ] )
  .then ( ([ res1 , res2 ]) => { console.log(">>" + res1 + "--" + res2 + "<<"); });

Promise.race( [ getUppercaseDataAfterDelay("abc",2000) ,
  getUppercaseDataAfterDelay("def",1500) ] )
  .then ( (firstResult) => { console.log(">>>" + firstResult + "<<<"); } );
```

>>>DEF<<<

>>ABC--DEF<<

6.6. Appel ajax via api fetch et Promise

L'api "**fetch**" supportée par certains navigateurs modernes utilise en interne l'api "Promise" de façon à déclencher des appels HTTP/ajax en mode GET ou POST ou autres.

Exemple en mode GET :

```
function myGenericJsGetFetchData(url){
  return new Promise((resolveWithJsData,reject)=>{
    fetch(url)
      .then( (response) => {
        if (response.status !== 200) {
          var errString = 'Problem. Status Code: ' + response.status;
          console.log(errString); reject(errString); return;
        }
        // Examine the text in the response :
        response.json().then(function(data) {
          resolveWithJsData(data);
        })
      })
      .catch((err) =>{ console.log('Fetch Error :-S', err); reject(err); });
  });
}
```

```
myGenericJsGetFetchData("/rest/produit/" + numProd)
  .then( (data) => { console.log(data);
    var jsonString = JSON.stringify(data);
    document.querySelector("#resProd").innerHTML = jsonString;
  })
  .catch((err) => { console.log(err); });
```

Exemple partiel en mode POST :

```
fetch(url,{ method: 'POST' ,
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body : JSON.stringify(jsObj)
})
  .then( (response) => {
    if (response.status !== 200) {
      var errString = 'Problem. Status Code: ' + response.status;
      console.log(errString); return;
    }
    response.json().then(function(data) {console.log(JSON.stringify(data));})
  })
  .catch((err) =>{ console.log('Fetch Error :-S', err); });
```


VII - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

https://fr.reactjs.org/	
https://www.taniarascia.com/getting-started-with-react/	

2. TP