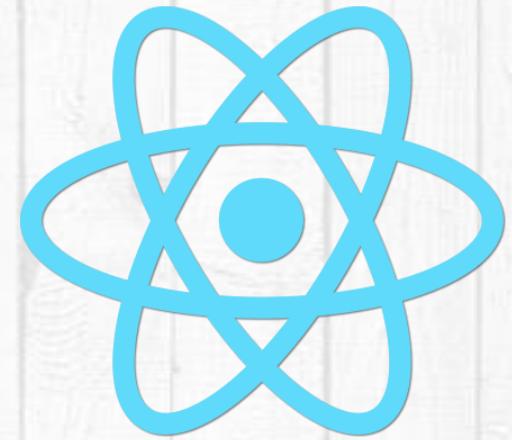


**ORSYS**  
formation

React.js



React.js



présentation logistique

## ORSYS & VOUS



# Bienvenue chez Orsys

## Présentation



- Indépendant
  - Prestataire d'Orsys
- Animateur
  - Monsieur DESORBAIX Alexandre
- Développeur
  - software, client lourd ,serveur
    - » c, c++, c#, java, ...
  - web, client léger
    - » html, php, sql, js, css, angular, react, ...
- blogger sous Wordpress



Bienvenue chez Orsys

logistique



- Nombre de jours :
  - 3 jours
- Horaires
  - Début : 9h
  - Fin : 17h30 approx.
- Pause
  - 10h30 approx. Matin
  - 12h30 approx. Midi
  - 15h30 approx. Après-midi



# PRÉSENTATION



- React c'est :
  - Un LIBRAIRIE != framework
  - Un ensemble de commandes d'assemblage, de test, ...
    - Une configuration automatique de l'environnement de dev
  - Un écosystème
  - Une communauté



# Une librairie

- React c'est une **librairie js/ts**, elle permet
  - De créer des applications clientes web
    - Créer de petit composant à assembler entre eux
    - Assembler de plus gros composants constituer de plus petit composants
    - Faire interagir ces gros composants entre eux
    - Créer une app grâce aux composants



Et c'est  
tout !!!!



# Une librairie

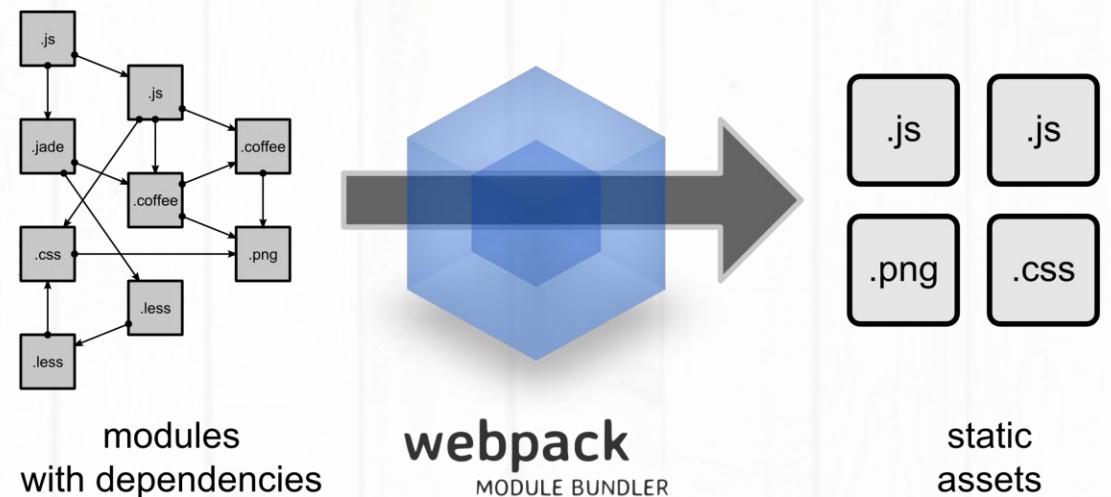
- React c'est une librairie js, qui permet aussi :
  - La mise en œuvre de tests
    - Reconfiguration de la plateforme d'automatisation des tests
  - La prise en charge de js et ts (webpack/babel)
- La librairie contient aussi une série de script serveur de construction d'app
  - Une commande **node.js** pour construire le squelette d'un app.



# Pour assurer la construction

- Webpack

- L'assembleur de module
  - Ex Avec common.js → require("")
  - avec `import bar from './bar'`
- React/cli nous fourni un environnement webpack préétabli



- Babel.js

- Se dit :le « compilateur » de js
  - Il compile le code et peut le rendre dans des Versions ES antérieurs
  - Transpile aussi le ts en js

BABEL



# Une commande

\*(option  
fortement  
recommandé)

- React est fourni avec \*une commande pour manager
  - la création d'objets diverses
  - Gérer les taches du projet
- Elle permet :
  - La construction d'un app (scaffolding) avec un modèle déjà défini.
    - Configuration de webpack
    - Définition de la structure de fichier

La génération(scaffolding) des :

- » Modules
- » Services
- » Composants
- » ,...

- Adeptes d'angular
  - NON
- React ne possède pas de fonction de génération
- Generate react cli le fait (\*;\*)
- <https://github.com/arminbro/generate-react-cli>



- Create-react-app
  - Scaffolding d'une appli
    - Dissociation code react / static asserts
    - Building
    - Serve
    - Webpack, préconfigurer
    - Ejection possible de CRA
  - Commande
    - `npx create-react-app mon-app`



# CONFIGURATION DU POSTE



# Windows Apache Mysql Php

## WampServer

- Gratuit, pour le dev rapide à mettre en œuvre



- Mdp root mysql: «»

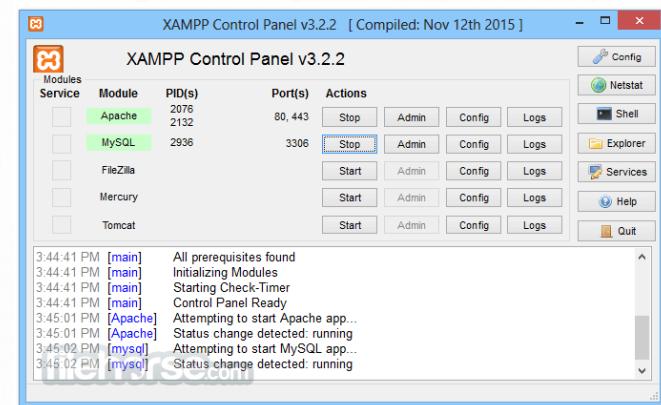


## XAMPP

- Gratuit, pour le dev rapide à mettre en œuvre
- Multiplateformes (linux osx win)



- Mdp root mysql: «»



# Windows Apache Mysql Php

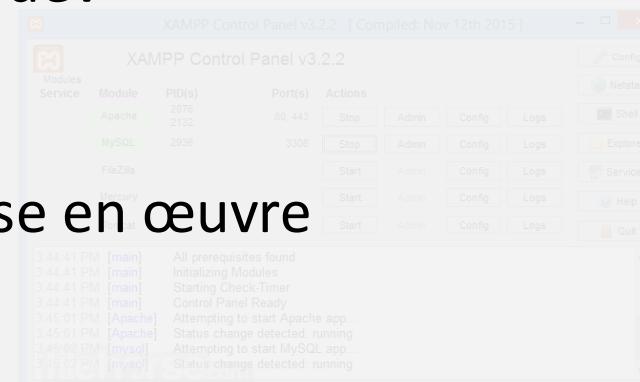
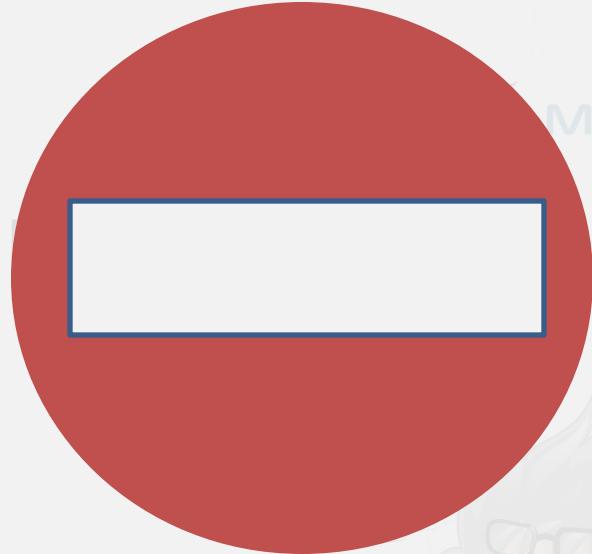
WampServer



XAMPP

- Gratuit, pour le dev rapide à mettre en œuvre
- Gratuit, pour le dev rapide à mettre en œuvre

- Seule une version builder pourrait avoir un sens à être servie avec apache
  - La couche php ne sera pas intéressante pour nous
- Pour le développement react nécessite vraiment l'outil de service pour le dev
- Sans Node js :
  - Complexité de mise en œuvre
    - Dev
    - test



# Choix d'un serveur

- Servir du html en production :
  - IIs
  - Apache
  - Nginx, node, ...
- Dynamisation
  - Coté client js
    - iis, apache, node, ...
  - Coté serveur js (code isomorphique)
    - Node
- Scafolder
  - create-react-app
    - Node.js
  - generate-react-cli
    - Node.js, ....

- roles
- servir les fichier
  - Gérer le js coté serveur \*option
  - Outils de dev



# Node.js & npm

- Node
  - Exécution de code JS côté serveur
    - Application
    - Scripting, ...
  - Lot d'applications existant
  - En ligne de commande
- Npm
  - Téléchargement de modules js
  - Gestion des dépendances



Utilise dans le  
cours pour les  
serveurs



- Commandes npm

- Initialisation d'un répertoire de projet
  - Créer un fichier minimal package.js
  - » **npm init -y**
    - Pré-configure node avec le répertoire .git (si .git présent)
    - -y : initialisation « quiet »
- Installation des paquets
  - » **npm install -g packetname packet2@version**
    - -g : installation pour l'utilisateur courant,
    - sans -g installation pour le projet, dans le répertoire node\_modules
    - --save permet l'ajout du packet aux dépendances du projet
- Npm start vs npm run monscript
  - Dans package.json, section scripts
  - Script name spécifiques nécessitant pas 'run' ex npm test, npm install, ...

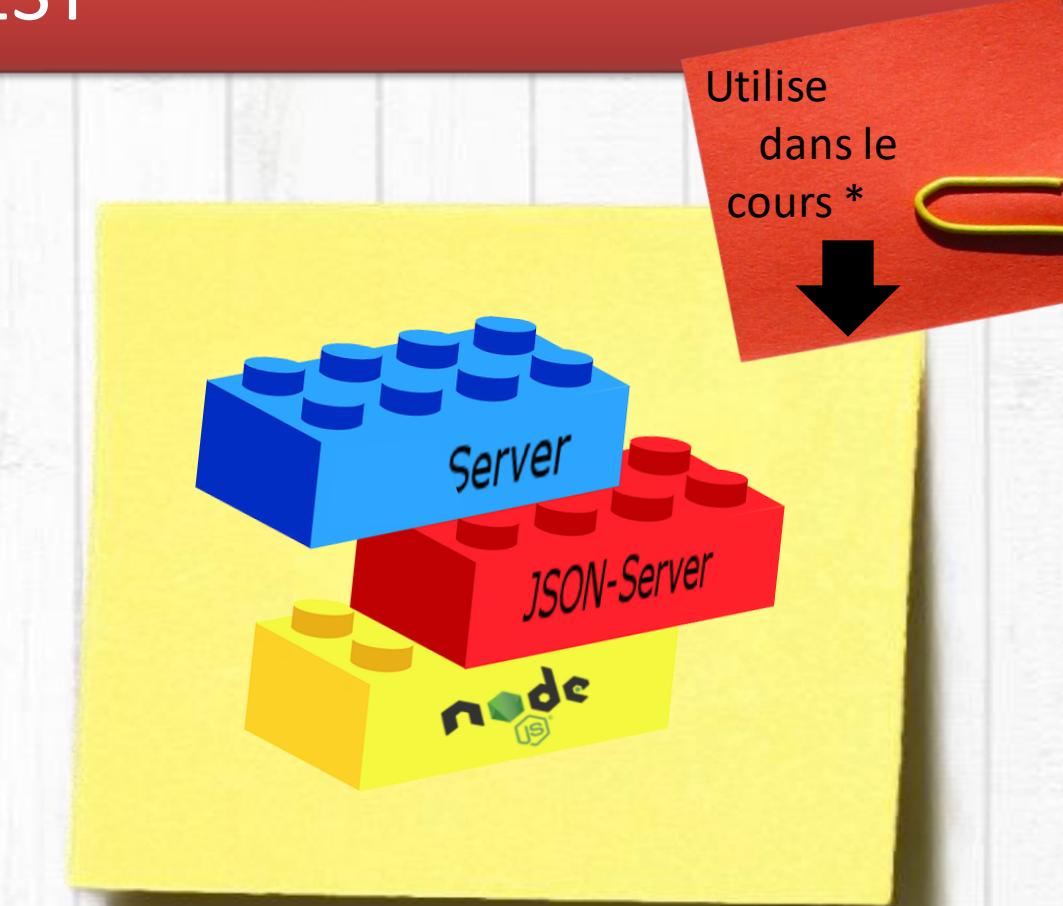


Il existe deux types de dépendances :  
devDependencies  
dependencies



# Serveur fake REST

- Json-server
  - Ecrit en node.js
  - Fake REST API
  - Mockeur de services rest
  - Stockage json
    - Création de endpoint en fonction des ressources du fichier
    - Ecriture direct sur le fichier json
  - <https://github.com/typicode/json-server>
    - npm install –g json-server



# Editeurs avancés pour le html/css/php/mysql/...

- Gratuit, multi langues, plugins & snippets, liaison debugger
- support GIT

## VS Code

angular-route.js — angular — Visual Studio Code

```
1 /**
2  * @license AngularJS v1.5.11
3  * (c) 2010-2017 Google, Inc. http://angularjs.org
4  * License: MIT
5  */
6 (function(window, angular) {'use strict';
7
8 /* global shallowCopy: true */
9
10 /**
11 * Creates a shallow copy of an object, an array or a primitive.
12 *
13 * Assumes that there are no proto properties for objects.
14 */
15 function shallowCopy(src, dst) {
16   if (isArray(src)) {
17     dst = dst || [];
18
19     for (var i = 0, ii = src.length; i < ii; i++) {
20       dst[i] = src[i];
21     }
22   } else if (isObject(src)) {
23     dst = dst || {};
24
25     for (var key in src) {
26       if (!key.charAt(0) === '$' && key.charAt(1) === '$') {
27         dst[key] = src[key];
28       }
29     }
30   }
31 }
```

Fichier Modifier Sélection Afficher Accéder Déboguer Aide

package.json AuthToken.js app.js angular-route.js ngSecure.js

Utilise dans le cours

SQL Pipelines (not connected) Li 17, Col 21 Espaces : 2 UTF-8 LF JavaScript

## Sublime Text 3

- Gratuit, multi langues, plugins & snippets, liaison debugger

C:\Users\Alex\Desktop\angular.js (workspace-node, www, ball, IoT-Building-Arduino-based-Projects-master, thing...) — Sublime Text 3

OPEN FILES

- finalSecurityExcl
- package.json
- json-server.php
- angular.min.js
- angular.js
- db.json
- index.html
- product.js
- products.js
- productsService.js
- Find Results
- product.html

FOLDERS

- workspace-
- FAN-tp
- ng-lamp
- .tmp
- app
- image
- script
- cont
- app
- other

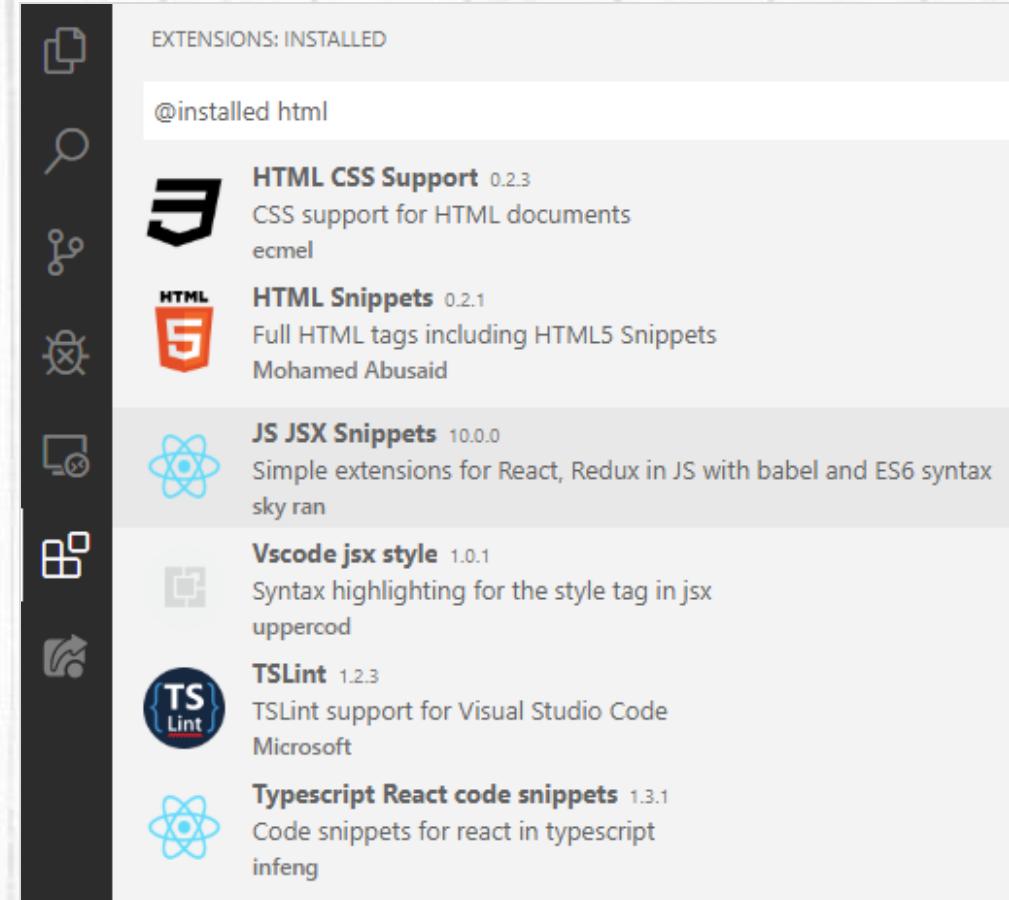
```
34 * error from returned function, for cases when a particular type
35 * @returns {function(code:string, template:string, ...templateArgs)}
36 */
37
38 function minErr(module, ErrorConstructor) {
39   ErrorConstructor = ErrorConstructor || Error;
40   return function() {
41     var SKIP_INDEXES = 2;
42
43     var templateArgs = arguments,
44       code = templateArgs[0],
45       message = '[' + (module ? module + ':' : '') + code + ']',
46       template = templateArgs[1],
47       paramPrefix, i;
48
49     message += template.replace(/\{\d+\}/g, function(match) {
50       var index = match.slice(1, -1),
51           shiftedIndex = index + SKIP_INDEXES;
52
53       if (shiftedIndex < templateArgs.length) {
54         return toDebugString(templateArgs[shiftedIndex]);
55       }
56
57       return match;
58     });
59
60     message += '\nhttp://errors.angularjs.org/1.5.6/' +
```

Line 55, Column 3 18% Spaces: 2 JavaScript



# Vs code les plugins

- Css3 support
  - Ajout de compléction
- Html support
  - Ajout de compléction html
- Js jsx snippet
  - Snippets
  - esLint
- Typescript react code snippet
  - Snippets
  - TSLint



# TypeScript react snippets

tsrcc → class component skeleton

tsrcfull → class component skeleton with Props, State, and constructor

tsrcjc → class component skeleton without import and default export lines

tsrpcc → class purecomponent skeleton

tsrpcjc → class purecomponent without import and default export lines

tsrpfcc → pure function component skeleton

tsdrpfcc → stateless functional component with default export

tsrsfc → stateless functional component

conc → class default constructor with props and context

cwm → componentWillMount method

ren → render method

cdm → componentDidMount method

cwrp → componentWillReceiveProps method

scu → shouldComponentUpdate method

cwu → componentWillUpdate method

cdu → componentDidUpdate method

cwum → componentWillUnmount method

gdsfp → getDerivedStateFromProps method

gsbu → getSnapshotBeforeUpdate method

sst → this.setState with object as parameter

bnd → binds the this of method inside the constructor

met → simple method

tscntr → react redux container skeleton

imt → create a import



# Serveur GIT

- Plateforme de disposition de code versionné
  - Repository
  - Gestion des commit distant
  - Travail d'équipe
- Plateformes gratuite
  - Github, GitLab
    - Hébergé ou officiel
    - 1 repo / account
  - Altasian BitBuckets \*(Sélectionné pour le cours)
    - Multi-repository
    - Free (max 1Go/repo) ou payed account
    - Officiel uniquement



GitLab



Utilise  
dans le  
cours



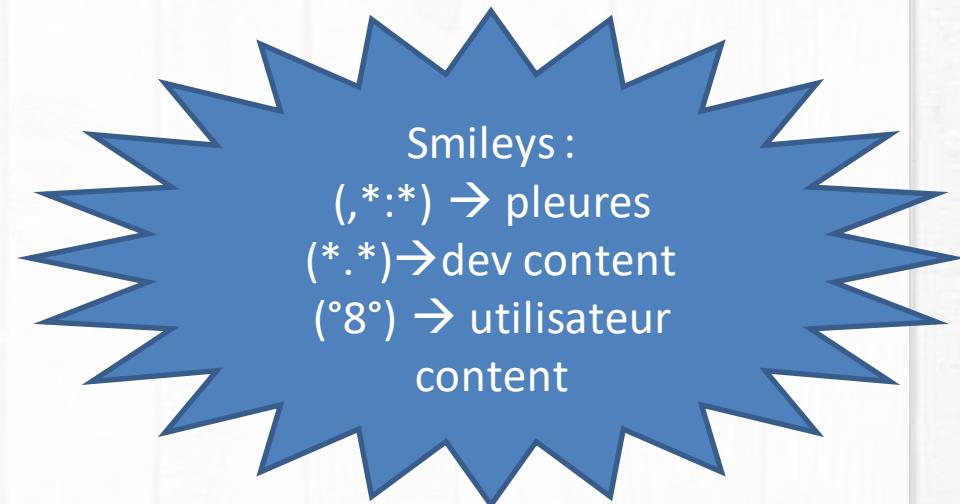
# Git - bitbuckets

- Création d'un compte gratuit
  - <https://bitbucket.org/account/signup/>
  - <https://bitbucket.org/account/signin/>
- Création 1ere repository
  - Edit, commit, approuve du readme.md
  - Clone local de la repository
- Ouverture du répertoire sous Visual Studio Code
  - Edition du readme.md sou vscode
  - 1<sup>er</sup> commit & push
- Approuve commit



# Outils de debug

- Tous les nouveaux navigateur ont leur console JS et leur outils de debug, avec possibilités de "*pas à pas*", *d'inspection*, *de watch*, ...
  - Chrome
  - Firefox
  - Edge
- En js des instructions permettent le debug dans la console
  - alert("message a affiché") *(,:\*)*
  - console.log(maVariable) *(\*.\* )*
  - modale *(\*.\* ),(°8°)*



# Configuration Node.js

- Node js
  - Moteur de code js côté serveur
  - npm
    - Gestionnaire de dépendances principale de node.js
  - Yarn
    - Gestionnaire de dépendances secondaire pour node



# RAPPELS ECMA / TYPESCRIPT



# Bonnes pratiques

- Les script chargées en derniers
  - <script src=« ... »></script></body>
- Mode strict
  - Permet d'exprimer des erreurs parfois sourdes ou juste des mauvaises pratiques
- L'usage de ‘...’ pour les chaînes de caractères dans le js
  - Var maChaine ='Demat breizh';
- Tous les blocs possèdent des { ... }
  - If( ... ){ ... }
- On limite l'usage des *eval()*
- On priviliege l'opérateur === ou !==
  - a === b
- On met les constantes en premier pour les comparaisons
  - if(undefined === maValue)



- ES 5 est succéder par différentes versions
  - Apportant de nouvelles fonctionnalités
    - Es6 (ou es2015)
    - puis es7
    - ,...
- Les navigateurs supportent à coup sur l'Es5 pas forcément les versions supérieures
  - Babylon.js convertit le code js dans différentes versions cibles



# Nouveautés syntaxiques es6 :

- L'usage du mot clef class pour la définition d'objets
  - Mot clef public, private\*(usage par #)
- Les arrow function
  - Fonction préservant le contexte (this) de déclaration lors de l'exécution
    - » En remplacement du .bind()
    - (arg)=>{ } eq. (function(arg){...}).bind(this)
- Les promise
  - Un promesse d'exécution grâce à then(()=>{})
- La nouvelle portée de scope : le module



# Nouveautés syntaxiques es6 :

- Les template strings

- Construction de chaine finale

- \${ expression } pour délimiter les évaluation de variables
    - Les évaluations peuvent s'imbriquer

- La chaine est définie avec accent grave et non des apostrophes

- ` une littérale `
    - ' ~~pas un littérale~~ '
    - " ~~pas encore une littérale~~ "

- Peut effectuer des choix, ternaires, ...

    ` \${chaines[0]}\${varI} `



- Speard operator
  - Permet de remplir un *array* ou *Iterable<T>* à avec les contenu d'un tableau b facilement

```
let tab1=[elem2,eleme3];
let tab2=[elem1, ...tab1]; //tab2 ➔ [elem1,elem2,elem3]
```
  - On pourrais imaginer la même chose pour des objets, ...
    - TypeScript l'a fait



# C'est l'histoire d'un mec ...

- Jean-Pierre se dit que sa chance vas tourner
  - Ce jour d'avril Il y avais une course de chevaux qui allais démarrer
  - Il a donc fais un pari sur un cheval
  - Il attend devant la course qui débute
  - Son cheval est placé
  - Il remporte sa mise



# C'est l'histoire d'un mec ...

- Jean-Pierre lorsque il boit son café au bar joue souvent au jeux d'argent.
- Plutôt chanceux, ce jour d'avril il se dit que c'est aujourd'hui

– Il gratte, il perd ..

– Il fais un loto

– Une course de chevaux, allais démarrer

- Il a donc fais un pari sur un cheval
  - Qui à un look de gagnant (cf photo 1).

- Il attend devant la course qui débute

- Son cheval est placé
  - Mais il a coché sans le vouloir le  
» 8 petit ange des prés (cf photo 2) → dernier
  - Enfin quand il arrivera, si il arrive

photo 1



photo 2

# C'est l'histoire d'un mec

- Jp sais qu'il a tout joué au tiercé du coup il rentre pu un sous en poche
- A 22h ce jour d'avril, il entend le tirage du loto
  - Souvenez vous, jp à joué avant le tierce :
    - Il avais fais sa grille  
3 – 8 – 21 – 35 – 49
    - Enregister son ticket contre un reçu de jeu
      - Il a donc prévu après le tirage de vérifier si il a gagner
        - » Il à déjà prévu que si il gagnait, il irais encaisser ses gains
          - et que c'est un gros lot, il déménagerais à Mulhouse
        - » Il a aussi prévu que si il gagnait rien, il jettérais son ticket
      - Vue qu'il étais d'humeur gagnante il a jouer et fait d'autre chose
        - C'est la notification de l'appli qui lui à rappeler que le tirage venait d'etre fait
    - Jp 42 ans en ce jour d'avril avais tous les bon numéro, dans l'ordre du loto.

Chanceux le type



# technique

- Le jeux a gratter n'avais aucune notion d'async
- La course de chevaux na pas été utiliser sous forme d'async
- Notion de promise es6 et **await** function es7/ts
- Le loto quand à lui :
  - Jouer une fonction **async** qui **await** le tirage du soir
  - Une fois que l'action de jouer est fini j' exécute une tache
    - En sachant à l'avance quelle prendrais du temps et que j'aurais bien d'autres taches à effectuer d'ici le résultat de la fonction appeler.
    - Je prévois directement, au moment de l'appel de la fonction, le **retour d'une promise**.
    - Promesse à laquelle je peux définir quoi faire quand l'exécution sera effectuer
      - avec     **.then(retour=>{ ... })**;



# Async / await / Promise

- Fichier.js

```
async function doSomething2() {  
    console.log(await doSomethingAsync())  
}
```

```
function doSomethingAsync() {  
    return new Promise((resolve) => {  
        setTimeout(() => resolve('I did'), 3000);  
    });  
}
```

```
async function doSomething() {  
    console.log(await doSomethingAsync())  
    console.log(await doSomething2())  
}
```

```
console.log('Before');  
doSomething();  
console.log('After');
```

- Pour être attendu un fonction :

- Doit soit être déclarer **async**

- Comporter un(des) **await**

- Doit retourner une promise

- Une fonction **async** s'utilise soit

- » Notion d'attente explicite, préfixer par **await**

- » Pas d'attente explicite, pas de préfixe à l'appel



# Les décorateurs

- Un wrapper pour les *High Order*
  - Le décorateur est abstrait, il n'est que syntaxe pour appeler une fonction
  - Créer un objet a de type A par une fonction B vivant dans la peau de B
- Disponibles de façon expérimentale depuis ES7
  - Intégré à type script
- Permet la création de décorateurs personnalisés
  - Définition d'une fonction a utiliser pour le décorateur



# Type script

- Du javascript en mieux
  - Un langage écrit par Microsoft
  - Sortie en version 0.8 en Octobre 2012
  - Dernière version 3.8.3 du 28 février 2020
- Proche du **js** et du **c#**, java
  - Cocréer par le principal inventeur du c# & concepteur du .NET
  - Interface, module, class, héritage, ...
- Un vrai langage Objets avec typage fort, intégration forte dans l'ide et dans node.js
- Un outils de trans-compilation
  - Du **ts** qui vers du **js** grâce à une commande « **tsc** »
- Un outil pour le ts
  - Un Linter « **tsLint** » permet le **marquage d'erreurs** et la **standardisation de l'écriture** du code



# Typage

- L'usage de **let** et **const** :
  - **let** : Son positionnement en mémoire peut changer d'objet mais rester du même type toute sa durée de vie.
  - **const** : le contenu référencé sera toujours le même, il ne pourra pas être réaffecté à un autre objet.
    - L'usage de **const** est bonne pratique dans maximum de cas, le linter peut vous le rappeler ...
- La portée de la variable est uniquement le **bloc** dans lequel elle est déclaré.
  - `( a: A )=> { let b = new A(); console.log(`a:${a} & b:${b} existe que dans cette fonction`); }`
  - `If( ... ) { const a = new A(); console.log(`a:${a} existe que dans cette fonction`); }`
- Typage basiques :

```
let varName: Type;  
let nombre: number;  
let chaine: string;
```
- Le Duck Typing

```
let departement = 56; //attention  
let fullName = `Sarah Vigote`;
```
- Rendre une variable facultative

```
let departement?: string; //attention aux accès
```



# Typage arrays et autre generiques

- Les génériques
  - Fonction nécessitant un type de quelle doit manager
  - Exemple d'un objet Array qui limite le type de contenu a des string
    - » Les fonctions de l'array vont definir de facon generique que si que des string sont stocker il peut renvoyer que des string
    - » **<T>** pour dire un type generique pas connus lors de l'ecriture d'une fonction
      - et dependra du type fournit lors de l'usage a chaque appel d'usage
- Le typage des *array* possède plusieurs syntaxes:
  - Il possède aussi les syntaxes suivante en plus de la syntaxe générifique
    - let names: Array<string>;**
    - let names string[];**
    - let names: [string];**



# Typage

- L'usage de let et const
- Typage basiques de variable:

```
let varName: Type ;  
let nombre: number ;  
let chaine: string ;
```

nom est un paramètre typés obligatoire

- Typer pour de fonction

```
function hello(
```

```
    nom: string,  
    age: number = 25,  
    isMale?: boolean  
): string
```

age est un paramètre typés obligatoire  
Mais avec une valeur par défaut

isMale est un paramètre facultatif  
sans valeur par défaut

La fonction doit  
renvoyer une string



```
{return `Demat ${nom} ${age}`;}
```



- Les classes
  - permettent le définition d'un objet
    - Des champs ou propriétés
    - Des méthodes du cycle de vie de l'objet
      - **constructor**,...
    - Des méthodes et des fonctions
    - Il peut avoir des champs qui lui appartienne qu'a lui
      - Notion de **private**, **protected**
    - Il peut avoir des champs/fonctions disponibles pour interagir avec lui
      - Notion de **public**



# C'est l'histoire d'un mec...

- C'est l'histoire d'un mec, de 56 ans habitant à mullhouse prénommé jean-pierre



- Il acheté une machine à café, le type ...
  - Elle à un bouton marche / arrêt
  - Un bac d'eau à remplir d'un litre
  - Et du café en poudre à mettre dans le filtre à café
- Analysons la conception de l'objet



# C'est l'histoire d'un mec...

- Sans intérêt pour nous

C'est l'histoire d'un mec, de 56 ans habitant à Mulhouse prénomme jean-pierre



- Identifions une class machine à café
  - Des fonctions d' interaction **public**
    - Elle à une fonction marche / arret
    - Un champs définissant le volume d'eau du bac
    - Elle possède un champ de type café
  - Des fonctions et champs qui lui sont propre
    - Température de l'eau
    - Activation de la résistance chauffante
    - Activation de la pompe
    - Eclairage du bouton lors du branchement
  - Une cafetière c'est donc un objet!!!  
Et jean-pierre aussi  
Heu l'autre!!



# Class machine à café

- Machine à café

- Des champs privés
- Des champs public
- Un constructeur
- Des fonctions interne privée
- Des interactions extérieur, public
  - This, context de l'objet dans l'objet

```
class MachineCafe{  
    private temperature: number;  
    private temperatureMax=95;  
    private isPumpStarted: boolean;  
    private isHeaterStarted: boolean;  
  
    public isOnButton: boolean;  
    public cafe?: Cafe;  
    public niveauEau: number;  
  
    constructor()  
    {  
        this.temperature=0;... /*  
        this.niveauEau=this.senseWaterLevel();  
    }  
  
    private senseWaterLevel(){}
    private startPump(){}
    private startHeater(){}
  
  
    public makeCofe(){  
        this.startHeater();  
        this.startPump();  
    }  
};
```

# C'est l'histoire d'un mec...

- Jean-pierre est pas adroit il met du café en poudre partout tous les matins



- Il acheté une machine à café,
  - **oui encore**, système **café à dosettes** cette fois
  - elle **fait tout pareil** ,mais plus pratique
    - Elle à un bouton marche / arrêt
    - Un bac d'eau à remplir d'un litre
    - Et un objet **filtre** qui contient **café** est **redéfini** différemment
      - Il accepte désormais du **café en dosette**
- Alors une machine à dosette c'est une machine qui fais aussi du café.

Comme l'autre!!!



# Class machine à café

- Machine à café à dosettes
  - Ses champs privés
    - Ceux du **parent** étant eux aussi **private** il sont **innaccessible**
      - » Notion d'**heritage**
      - » Notion de **protected**
  - Des champs / fonctions **public**, **protected**
    - Permettant l'héritage aux futurs extensions
    - **Redéfinition** de la manière de faire le café
      - » Notions d'**override**
  - Des champs / fonction privés qui lui sont totalement personnel
  - Un constructeur
    - Appel du constructeur parent
      - » Notion de **super()**
  - Des fonctions interne protégé pour permettre aux futurs génération de pleinement hérité de leur ainés

```
class CoffeMachinPad extends CoffeMachin{  
    public padHolder: PadHolder;  
  
    constructor()  
    {  
        super();  
    }  
  
    public insertDoset(pad:CoffePad){ ... }  
  
    public makeCofe(){  
        this.coffee= this.padHolder.coffee;  
        this.startHeater(97);  
        this.startPump(255);  
    }  
};
```

# C'est l'histoire d'un mec...

- Jean-Pierre est devenu un expert du café
  - Il a essayé une 20aine de systèmes différents.
- Il à finit par déduire que
  - Faire du café **c'est toujours :**
    - Mettre du café/ eau
    - Démarrer
    - Collecter boisson chaude
  - abstraction de faire le café
    - Les machines implémentent les fonctions de faire du café commun et possède des champs communs
    - Il peuvent grâce à **implements** bénéficier de l'abstraction d'autres fonctionnalités, ex : mousseur à lait



# C'est l'histoire d'un mec...

- Jean-Pierre est devenu un expert du café
  - Il a essayé une 20aine de cafés diffèrent.
- Il à finit par déduire que
  - Le café est toujours caractérisé de la même manière mais nécessite pas un besoin d'objet définit Seul le fait qu'il contienne tout me suffit
    - Il utilise donc une interface qui définit juste le contenu décrit
      - Arome,
      - Provenance,
      - ... mais **aucune fonctionnalité** a proprement parler
    - Il formalise juste qu'est ce qu'on peu appeler du café et qui le caractérise



# C'est l'histoire d'un mec...

- Jean-pierre est devenu un expert du café
  - Il a essayé une 100aine de cafés diffèrent.
- Il à finit par déduire que
  - Il a créer une classification pour les différents champs plutôt qu'un simple nombre
    - Il utilise donc une **enum** qui définit juste une catégorisations
      - Arome,
        - » corsé, doux
      - Provenance,
        - » Pérou, Mexique, ...
      - ... mais **aucune fonctionnalité** à proprement parler
    - Il formalise et catégorise pour une valeur



# La gestion de imports

- Avec **import et export / default / import {} from**
- Syntaxe :
  - pour être importer, un contenu doit être exporter

```
export class ExportedThing{};  
export default class DefaultExported{...;}
```

- Puis importer a la demande

```
import {ExportedThing} from './rep/fichier';  
import DefaultExported from './rep/fichier';
```



- La phase de transpilation (ex: tsc) vérifie l'intégrité de forme des variables
  - Un fichier tsconfig.json qui spécifie :
    - Le type de sortie
    - Les fonctionnalités prendre en charge ,...
  - Il se génère aisément grâce à **tsc --init**
  - Attention cette phase de compilation garanti uniquement l'intégrité des types jusqu'à la transpilation
    - » Le code sera **exécuter** avec une version **js générer**



# Tscconfig.json

- Exemple de fichier tscconfig.json générée par tsc -init

```
{  
  "compilerOptions": {  
    /* Basic Options */  
    // "incremental": true, /* Enable incremental compilation */  
    "target": "es5", /* Specify ECMAScript target version: 'ES3' (default) */  
    "module": "commonjs", /* module code generation: 'none','commonjs','amd' */  
    // "lib": [], /* Specify library files to be included */  
    // "allowJs": true, /* Allow javascript files to be compiled. */  
    // "checkJs": true, /* Report errors in .js files. */  
    // "outFile": "./", /* Concatenate and emit output to single file. */  
    // "outDir": "./", /* Redirect output structure to the directory. */  
    // "rootDir": "./", /* Specify the root directory of input files. */  
    // "removeComments": true, /* Do not emit comments to output. */  
    // "noEmit": true, /* Do not emit outputs. */  
    // "isolatedModules": true, /* Transpile each file as a separate module */  
  
    /* Strict Type-Checking Options */  
    "strict": true, /* Enable all strict type-checking options. */  
  
    /* Experimental Options */  
    // "experimentalDecorators": true,/* Enables support for ES7 decorators. */  
    // "emitDecoratorMetadata": true, /* Enables support for emitting type of  
    // decorators. */  
  }  
}
```

Doc : <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>

- Une série de paramètres
  - La cible visée
  - La gestion des modules
  - Les chemins entrés / sorties
  - ...
- Les paramètres pour une grande partie peuvent être **set** dès la ligne de commande tsc

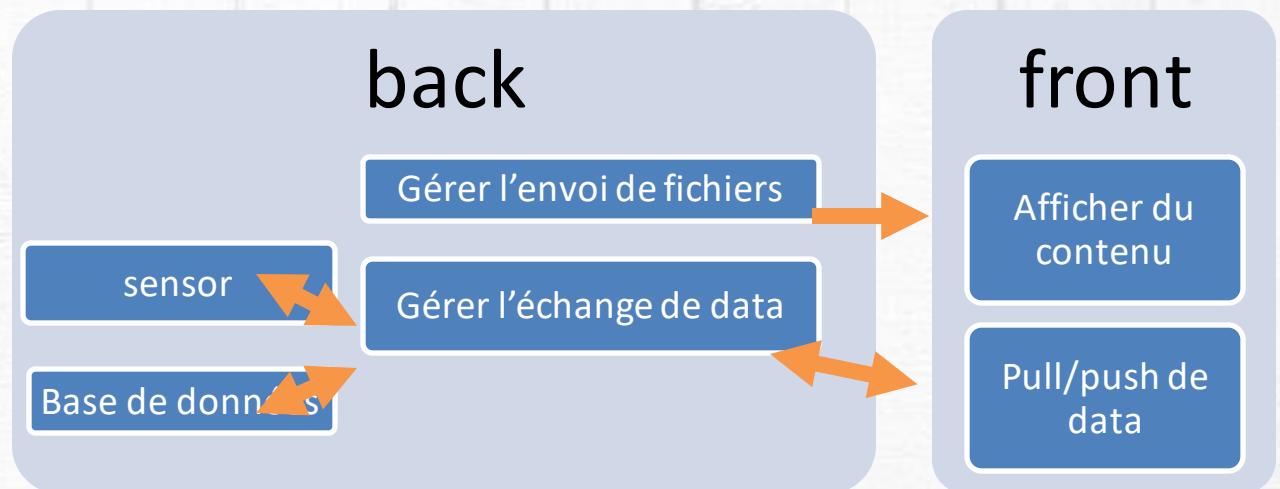


# RAPPELS ARCHITECTURE



# Qui fais quoi ?

- Qui à besoin de quoi ?
  - Le client, Un navigateur
    - Charge le site et met en action le JS
    - Il géra grâce au js les interactions IHM
    - Chargerà une version du site html/css/js
      - Et les librairies js associes
    - Il géra les échanges http
      - Accès à des données par Webservice (REST, json)
  - Le serveur
    - Il Répondra aux requêtes http pour accéder aux fichiers du site
    - Il répondra aux requêtes liées aux données (webservice tel que REST)
    - Il géra des échanges internes pour l'accès au sgbdr, aux sensors, ...



# Un web service

- Web Services :
  - client,
    - celui qui accède
  - fournisseur,
    - celui qui diffuse
  - annuaire de services
    - celui qui recense ex UDDI pour les WSDL \*(SOAP)
  - intégration d'applications
    - Des API
- Métaphore :

le serveur du restaurant et sa "carte du menu", on se sers pas directement en cuisine



# Un bistro service ?

- Un web service c'est comme  
Un bistrot

- Il est de bon ton d'adopter le protocole adéquat
  - Bonjour
- Un serveur prend ma commande
  - Une bière pression
  - L'hydromel du patron est une bouteille  
Qui ne peut être servis aux clients
- Le serveur me sers ma commande  
si cela est possible.
  - Je consomme ce qui m'a été servi dans
    - le verre adéquat
    - le volume parfait



# Un bistro service?

- Dans certaines enseignes il est possible de savoir ce qui est possible de commander
- Il est même expliquer que
  - Sur présentation d'un coupon Promo une réduction spéciale sera faite

| APÉRITIFS                                |      |        |
|------------------------------------------|------|--------|
| Pastis CABANEL                           | 2cl  | 3,50 € |
| Muscat de St Jean BARROUBIO              | 8cl  | 4,50 € |
| Vin de Blanquette 1 <sup>ère</sup> BULLE | 10cl | 5,50 € |
| Maury MAS AMIEL                          | 8cl  | 6,00 € |
| Martini Bianco                           | 8cl  | 5,00 € |
| -Carcassonne- CABANEL                    | 10cl | 4,00 € |
| Ricard                                   | 2cl  | 4,00 € |
| Vin de Vin du moment                     | 15cl | 4,00 € |
| Coupe de Champagne DEUTZ                 | 10cl | 9,00 € |
| L'apérolée                               | 10cl | 5,50 € |
| Aperol Spritz                            |      | 7,50 € |

## PLAT DU JOUR 13,50 EUROS UNIQUEMENT LE MIDI

### MARDI

Poulet fermier rôti et frites maison

### MERCREDI

Blanquette de veau et riz Pilaf

### JEUDI

Joue de boeuf braisée au vin rouge

### VENDREDI

Effiloché d'aile de Raie et pomme de terre écrasée. Emulsion huile d'olive et citron vert

### POT LYONNAIS 46 ET 25 CL

ROUGE Castelnaudary Cabrèras 7,50 € et 5,50 €  
BLANC Vacqueyras Pic St Loup 7,50 € et 5,50 €  
ROSE Jouclar Cabardès 7,50 € et 5,50 €

### LES EAUX MINÉRALES

VITTEL 0,5 litre 3,50€ 1 litre 5€  
PEYRE 0,5 litre 3,50€ 1 litre 5€  
SAN PELIGRINO 0,5 litre 3,50€ 1 litre 5€

### BIÈRE PRESSION

HEINEKEN 15 cl 2,50 €  
HEINEKEN 25 cl 3,50 €  
HEINEKEN 33 cl 4,50 €  
HEINEKEN 50 cl 7,00 €

### JUS DE FRUITS & SOFTS

COCA COLA - COCA ZERO 3,50 €  
ICE TEA 3,50 €  
LIMONADE 3,50 €  
JUS D'ORANGE PRESSÉ 5,50 €

### BOISSONS CHAUDES

Café expresso 2,00 €  
Thés et infusions 3,50 €  
Décaffiné 2,00 €  
Cappuccino 4,00 €



26 RUE CHARTRAN 11000 CARCASSONNE

### Pour commencer

|                                                                                       |         |
|---------------------------------------------------------------------------------------|---------|
| Palet de pied de cochon désossé et pané en salade                                     | 11,00 € |
| Salade d'endive (noix, cassis, tomates séchées, œuf mollet, crème fraîche et fromage) | 12,50 € |
| Hélices de canne d'Auragail                                                           | 9,50 €  |
| Saumon fumé par nos soins (au bois de hêtre), crème cibonne et blinis                 | 14,00 € |
| Toasts de canard à la confiture de figue                                              | 15,00 € |
| Emietté de poule de Méditerranée grillé et mousseline de parais                       | 18,00 € |

### Plats de résistance

|                                                                                                                                   |         |
|-----------------------------------------------------------------------------------------------------------------------------------|---------|
| Entrecôte «Aubrac» sauce Béarnaise                                                                                                | 25,00 € |
| Emincé de côte de veau du «Ségala», dauphinois de légumes et jus de veau à la truffe                                              | 23,00 € |
| Pâté fermier farci aux morilles, royale de champignons                                                                            | 20,00 € |
| Épaule d'agneau longuement confit au naturel, ficassée de pois cassés et jus au thym                                              | 19,00 € |
| Tartare de boeuf «Aubrac» préparé                                                                                                 | 17,00 € |
| Lapin rôti au romarin et à la moutarde à l'ancienne, frites maison et quelques feuilles de salade pour se donner bonne conscience | 16,00 € |

Toutes nos viandes rouges «Aubrac» sont accompagnées de frites maison et salade verte

### Poissons

Poisson du Jour au gré du Marché

Voir ardoise

Taxes et service compris - Viandes Origine France  
LA MAISON N'ACCÈPTE PAS LES CHÈQUES



- Définition

Un service Web est un programme informatique inter-opérable permettant la communication et l'échange de données sans intervention humaine et en temps réel (Wikipedia). Basé sur des standards et protocoles ouverts (cf XML)

- Dans la pratique

- Un service Web est une fonctionnalité (ou un ensemble de fonctionnalités)
- Mise à disposition via une interface (ressource) si possible unique
- Un service peut fournir :
  - du contenu (éventuellement dynamique) mis à disposition (ex: liste de produits)
  - une interface d'enregistrement (ajout, suppression, modification dans une base de données)
  - des fonctionnalités métier (ex: calculs de TVA par pays, des taxes d'une fiche de paye...)

- Les formes du service Web

- SOAP (WS-\*), REST (RESTFull), appel RPC (version plus ancienne)



# Un bistrot sur le web?

- Le bar symbolise:
  - Réserve
    - les données, sgbdr, sensor, ...
  - Serveur du bar
    - comprendre une demande
    - Vérifier que le protocole & la demande sont bien formulés
    - Servir, traiter, assembler en fonction de la demande
  - Le client, nous humain,
    - machine consommant ce qui est servie en fonction du context
  - Quand à la carte
    - On parle ici de couche de **définition** de ce qui est **mis à disposition** ainsi que le **formalisme** pour le demander



- REpresentational State Transfert
  - Transfert de représentation d'état
    - CRUD
  - Permet l'échange de données
    - Couple ressource / uri
    - Appelé *endpoint*
  - S'appuie fortement sur le protocole HTTP
  - Transfert de ressource xml ou json (json est privilégié pour le js)

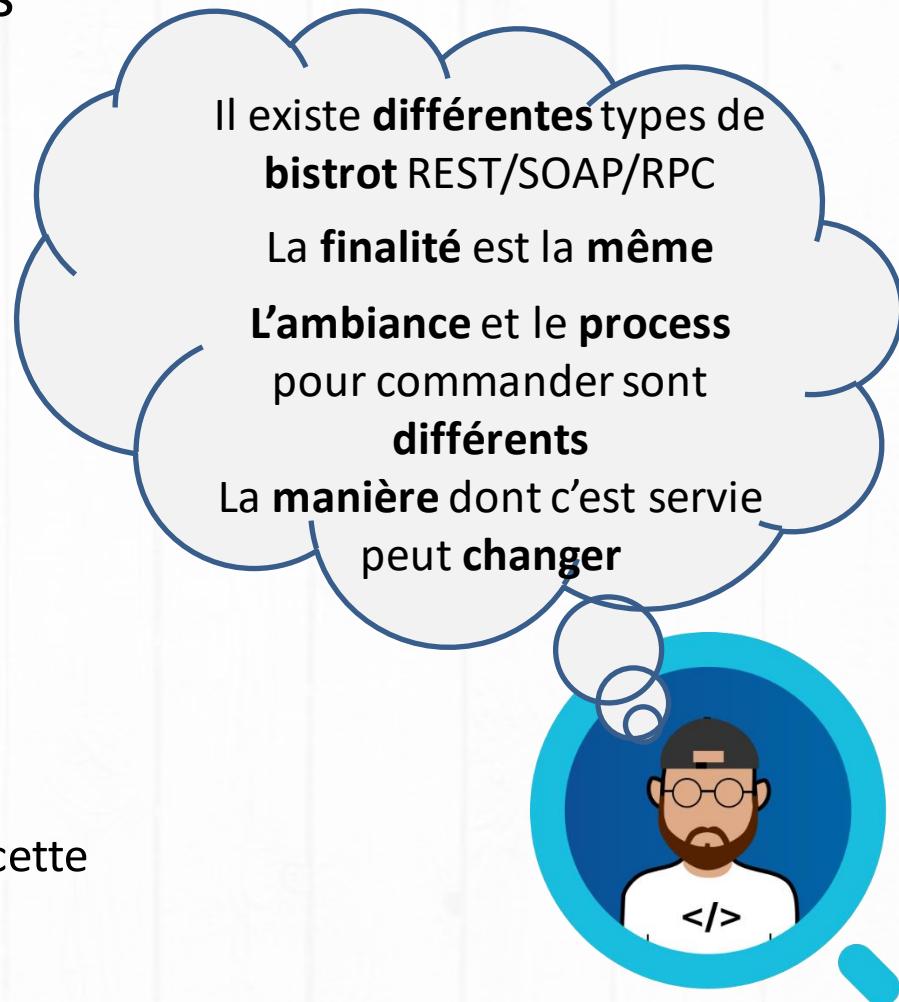


- Accès CRUD
  - Create
  - Read
  - Update
  - Delete
- Accès à une ressource par HTTP
  - Méthodes HTTP
    - POST (create)
    - GET (read)
    - PUT (update)
    - DELETE (delete)
  - Capacités
    - Filtrage
      - champs
      - id
    - Positionnement
    - Relationnel \*
  - Erreurs HTTP
    - 2xx OK
    - 4xx Erreur accès
    - 5xx Erreur Serveur



# Un bistrot sur le web !

- On parle **ici** de deux type de webservices
  - SOAP
  - REST
- La réserve
  - Votre partie applicatif qui
  - construit les résultats
  - Accède au serveurs internes
  - Le moteur du web service
  - Effectue les contrôles
- Le serveur du bar
  - Le logiciel serveur qui reçoit les requêtes
  - Redirige les requêtes vers la réserve
- La carte
  - On parle **seulement** ici du **SOAP** qui permet cette logique de définition
  - WSDL (WebServiceDefinitionLayer)



# Tout le reste n'est que JS

- React ne permet que la confection rapide de l'app
  - Assembler avec des component
- Qui s'occupera du **rest** ?
  - Les appels http
  - Le cycle de vie
    - Le composant est déjà monté
    - **componentDidMount**
      - Initial fetch for a component and son
  - await / async (es7/ts)
    - **async componentDidMount**



# l'intégration de rest WS en js

- Pour faciliter l'accès aux données nous privilégierons le rest
  - Les échanges XML sont plus durs à mettre en œuvre et demande plus de ressources que je le JSON
- En js Historiquement les appelles http(sync et async) étaient fait par xhr (ou XMLHttpRequest)
  - **Xhr**
    - Async grâce à onreadystatechange
    - Prise en charge : fortement compatible
  - **Fetch** Une nouvelle approche est disponible dès es6
    - Approche async promise avec then()
    - Prise en charge : moins compatible

Compatibilité Navigateurs

|                            | PC     |      |         |                   |       |        |                 | Mobile              |                      |                    |                |                  |
|----------------------------|--------|------|---------|-------------------|-------|--------|-----------------|---------------------|----------------------|--------------------|----------------|------------------|
|                            | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome pour Android | Firefox pour Android | Opera pour Android | Safari sur iOS | Samsung Internet |
| fetch                      | 42     | 14   | 39      | Non               | 29    | 10.1   | 42              | 42                  | 39                   | 29                 | 10.3           | 4.0              |
| Support for blob and data: | 48     | 79   | ?       | Non               | ?     | ?      | 43              | 48                  | ?                    | ?                  | ?              | 5.0              |
| referrerPolicy             | 52     | 79   | 52      | Non               | 39    | 11.1   | 52              | 52                  | 52                   | 41                 | Non            | 6.0              |
| signal                     | 66     | 16   | 57      | Non               | 53    | 11.1   | 66              | 66                  | 57                   | 47                 | 11.3           | 9.0              |
| Streaming response body    | 43     | 14   | Oui     | Non               | 29    | 10.1   | 43              | 43                  | Non                  | Non                | 10.3           | 4.0              |



# Examinons fetch

- Method GET par defaut
  - 1<sup>er</sup> then : pour convertir un stream de réponse  
→ réponse lisible
    - Différents formats
    - Conversion json ->objet
  - 2eme then : traitement après conversion
  - Un objet de configuration de la demande peut être fournit en 2eme arg de fetch
    - `fetch('https://www.reddit.com/r/javascript/.json', {  
 method: 'post',  
 body: JSON.stringify(opts)  
}).then(...).then(...)`

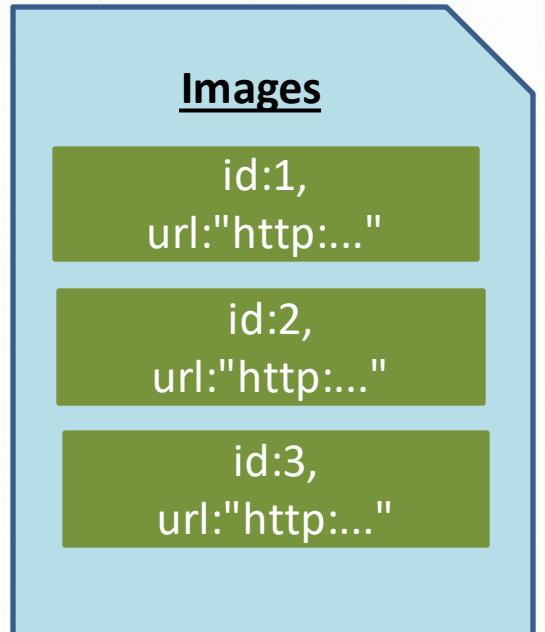
- Exemple GET :

```
const myImage =  
document.querySelector('img');  
fetch('flowers.jpg')  
.then(function(response) {  
    return response.blob();  
})  
.then(function(myBlob) {  
    const objectURL =  
URL.createObjectURL(myBlob);  
myImage.src = objectURL;  
});
```



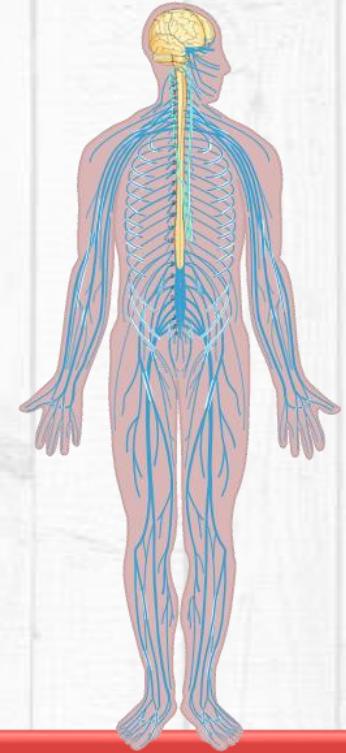
# TP rappels

- Exécuter un serveur json-serveur
  - Sur le flux en commentaire
    - <http://mon-json-server/images> et <http://mon-json-server/images/1>
- Faites un **objet** permettant,
  - d'effectuer des appels Rest pour
    - GET une ressource images sur le json-server
    - GET sur un ressource générique sur le json-server
  - Exécuter une call une fois la réponse reçue
    - Console log de l'objet reçue
  - Prévoir aussi les fonctions pour POST, PUT, PATCH, DELETE



Y'a quoi dedans ?

## CONCEPT REACT



- Create-react-app
  - Scaffolding d'une appli
    - Dissociation code react / static asserts
    - Building
    - Serve
    - Webpack, pré-configurer
    - Ejection possible de CRA
  - Commande
    - `npx create-react-app mon-app`



- Create-react-native-app
  - Scafolding d'une appli react-nativeBuilding
    - Serve
    - Compilation & Simulation sur android (win & mac)
    - Compilation et simulation ios (xcode mac)
    - Ejection possible de CRNA
  - Commande
    - `npx create-react-native-app mon-app`



# Mise en place d'un appli

- Exécuter la commande `create react app`

`npx create-react-app app`

- Installation automatique de dépendances :
  - Cra-template, modèle type de structure d'app react(structure de fichiers)
  - Babel et configuration
  - React script(build,serve, ...)
  - React dom (react.createElement(),...)
- Cra-template sera supprimer une fois l'app créer

info Direct dependencies

  |  
  | cra-template@1.0.3  
  | react-dom@16.13.1  
  | react-scripts@3.4.1  
  └ react@16.13.1

info All dependencies

  |  
  | @babel/plugin-transform-runtime@7.9.0  
  | babel-preset-react-app@9.1.2  
  | cra-template@1.0.3  
  | eslint-config-react-app@5.2.1  
  | react-dev-utils@10.2.1  
  | react-dom@16.13.1  
  | react-error-overlay@6.0.7  
  | react-scripts@3.4.1  
  | react@16.13.1  
  └ scheduler@0.19.1

yarn remove v1.21.1

[1/2] Removing module cra-template...

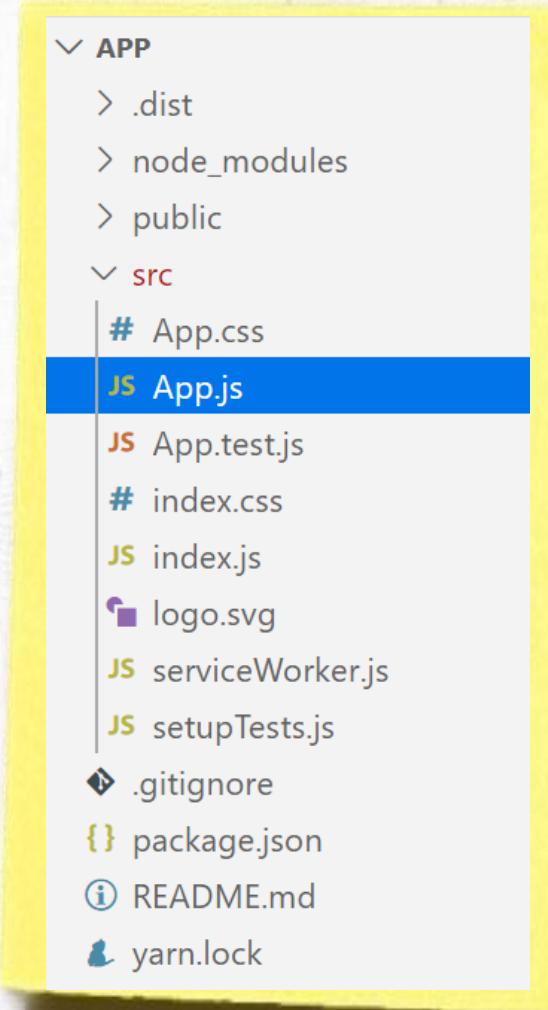
# Cra scripts commandes

- Start
  - **Version de dev**
    - Service http pour le code de dev
    - Outils de debug
    - Websync
- Build
  - Build la version minifier du code source
  - **Version de production**
- Test
  - Exécute les commandes de test de l'appli



# Arborescence

- **folders**
  - **node\_modules**
    - Modules nodes (nécessaire à node js, create react app et autres lib, ...)
  - **public**
    - Static assets
    - Contenu public constant , image , icônes, ...
  - **src**
    - Code js / css / scss / jsx / ts/tsx...



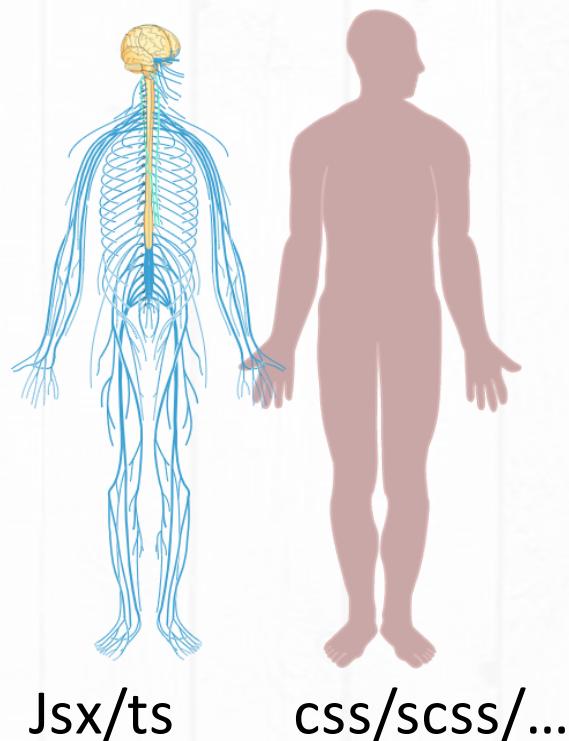
# Analysons le répertoire src

- Le répertoire est constitué de couples de fichiers portant le même préfixe (ex: App):

- .scss/.css
  - » Description du comportement
  - css du composant
- .js / .jsx / .tsx\* / .ts\*

  - » Le code react du composant

- .test.js
  - » Description d'un composant
  - » Un contrôle que le composant se créer correctement



- Structure générée par CRA pour le répertoire src

```
✓ src
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
```

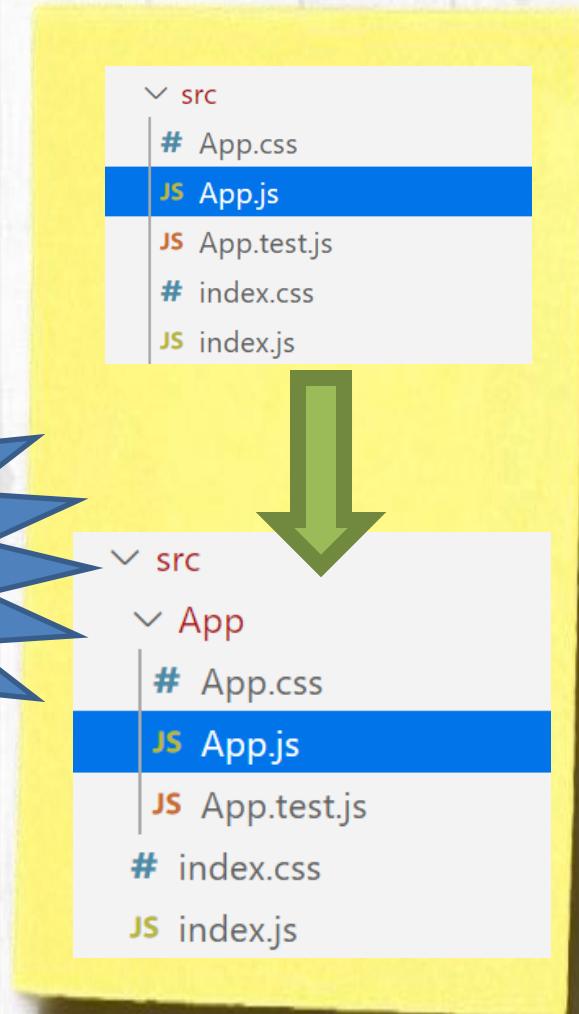


# Analysons le répertoire src

- Structurer correctement votre app

- Notion de découpage de component
  - Feature component
  - UI component
  - Application component
- 1 Répertoire = 1 component
  - Js +css
  - Si besoin un / plusieurs enfant Dans un répertoire pour chaque
- 1 répertoire pour chaque catégories de component

Le plus important  
c'est les règles  
communes aux  
projets  
**Dev guide lines**



# Component concepts

- Un component
  - Reçoit des props  
(en lecture seul)
  - Commence toujours par une majuscule
    - Pour dissocier les Composant et les html
  - Gère son cycle de vie
  - Il existe 2 types de component
    - Les fonctions, composants simple exprimant des valeurs
    - Les objets Class, composants plus complexe et plus lourd, gérant plus d'interactions



Je m' appelle  
Alexandre  
pas  
**alexandre**

Chaque étape de  
ma vie est déjà  
définit



# Component Function vs class

## fonction

- Reçoit des props
  - Dans l'argument d'entrée de la fonction
- Sans état interne
  - Stateless *ou* gestion Hook
- **return** le contenu du composant assemblé avec ses props
- Facile à tester
  - La fonction s'exécute et on doit forcément pouvoir prédire la forme du résultat

## class

- Reçoit des props
  - Dans le constructeur (**constructor**) de la class
- Avec ou sans états interne
  - Statefull/stateless
  - `this.setState()`, `this.state`
- Possède une fonction **render()** qui retourne le contenu du composant assemblé avec props & état(**state**)
- Extension de class
- Test plus lourd mettre en œuvre



# Import & export Component

- Importer des composants
  - Js
    - Import DESCRIPTEUR from './folder/file'
  - CSS ,...
    - Import './fichier.css'
- Export par default
  - Obligatoire pour react
  - Export default MonComposant



# Créer votre premier composant



- Création du composant
  - Usage dans l'app
  - Passage de props
    - Affichage des props
  - Faire un composant qui décompte le temps
    - De 100 à 0
    - Essaie avec generator react cli pour le component



# Evénements

- Les évènements sont des évènements javascript donc ils se déclarent par leurs noms JS

📄 ActionLink.js

```
function ActionLink(props) {
  function handleClick() {
    preventDefault();
    console.log('Le lien a été cliqué.', props);
  }
  return ( <a href="#" onClick={handleClick}>{props.value}</a> );
}
```

- onClick={monHandleClick}
- onChange={handleChange}
- ...

- Les events onClick, onChange, ... existent que sur les éléments html.
- Sur vos component à vous de les implanter en
  - envoyant par les props votre fonction d'évènement,
  - la faisant exécuter à l'exécution de l'évènement voulu dans le component



# Jsx subset

- Js + XML dans le même fichier

Permet le retour de contenu de composants

- Blocs XML Commencent et finissent par ( )
- Dans ces blocs xml,
  - les commentaires s'écrivent :
    - `{/*comm.*/}`
  - Les sections en js dans un partie html de jsx
    - `{ monarray.map(e=>{<balise/>} ) }`



```
function App(props) {
  return (
    <div className="App" >
      /* parcour d'array de logos */
      {
        props.logos.map(
          e=>{<img src={logo}>})
      }
    </div>
  );
}
export default App;
```



- Js + XML dans le même fichier
  - **Attributs** xhtml disponibles par leurs noms JS
    - Ex :
      - » Id →
      - » style →style
      - » class → className
  - L'attribut **style** est un objet en js et reçoit donc un objet de styles css
    - Ex : style={{textAlign:'center', fontWeight : '900'}}}



# Usage des composants

- Les composants sont utilisables sous forme de balises

📄 Composant.jsx

```
Import MonComponent from './moncomponent/moncomponent'  
function(props){  
    return (<MonComponent valeur1={Objet/funct} valeur2="valeur"/>);  
}
```

Les **balise** portent le **nom** du **composant** qui doit être insérer

Il **reçoit ses props sous forme d'attributs** de balises

Chaque attribut créera un champs dans les props  
Peut recevoir des valeur, des objets des fonctions, ...



Il doit être **importer / exporter correctement** et connu lors de la présence de la balise



# PropsType

- Contrôle du type de contenu pour les props :
  - `propsType`
  - `isRequired`
    - Sur tous les types
    - **Rend obligatoire, si pas présent = facultatif**
  - Types existant :
    - `string`
    - `number`
    - `object`
    - `Array`
    - `Bool`
    - `Func, ...`

Liste exhaustive :  
<https://fr.reactjs.org/docs/typechecking-with-proptypes.html#proptypes>

## Composant.js

```
import PropTypes from 'prop-types';
```

```
class Greeting extends React.Component { ... }
```

```
Greeting.propTypes = {  
  name: PropTypes.string  
};
```



# defaultPropsType

- Contrôle du contenu des props :
  - Positionner des valeurs si elles ne sont pas fournies dans les props lors de la création du composant <MonCmp .../>
  - Même fonctionnement que propsType mais avec des valeurs, au lieu des types

## Composant.js

```
Greeting.defaultProps = {  
    name: 'nom inconnu'  
};
```



# State et data dynamique du component

- Le state est un conteneur pour les variables du composant
- Il est initialisé avec un état initial
  - Mickael beau jeune homme afro américain
- Il est modifier par le composant ou autre,
  - Mickael à subit des changements de son state
    - On parle de l'état de Mickael à cette instant
    - Son render a été affecté
  - Mickael à re subit un changement de son state
    - On parle du nouvel état issue de l'état initial et ayant subit déjà un changement



# Class et state

- Le state doit être initialiser dès la création

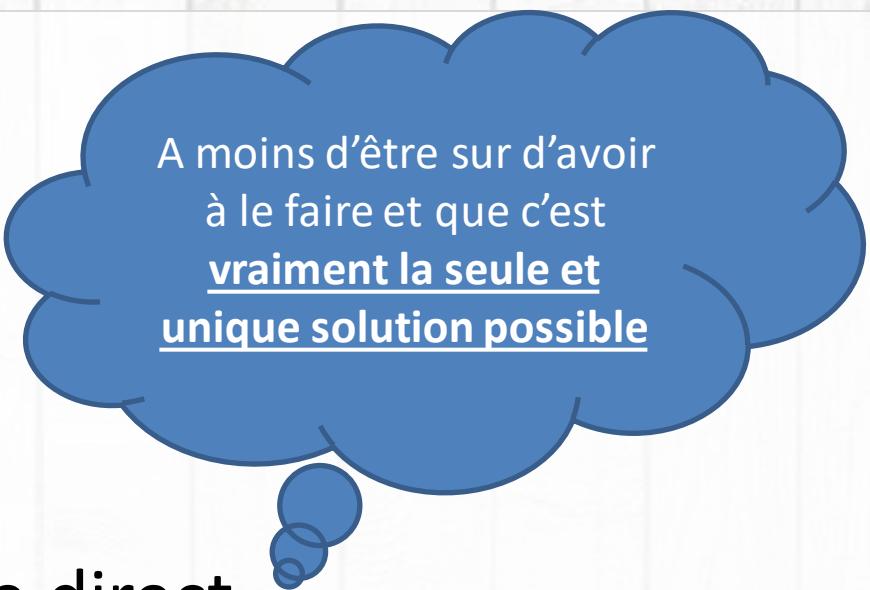
- Dans le constructeur :

📄 Cmp.jsx

```
Class Cmp extends React.Component{  
  constructor(){  
    this.state={ contenu de départ };  
  }  
}
```

- Le state doit jamais être accédé en écriture direct

- Lors d'écriture sur le state du composant utiliser `this.setState({})`
  - Gestion async du changement
    - Déclenchement des fonction de cycle de vie (`shouldComponentUpdate`)
  - Déclenchement du **render()** en fin de modification du state
    - Cascade de mise à jour des props des enfants



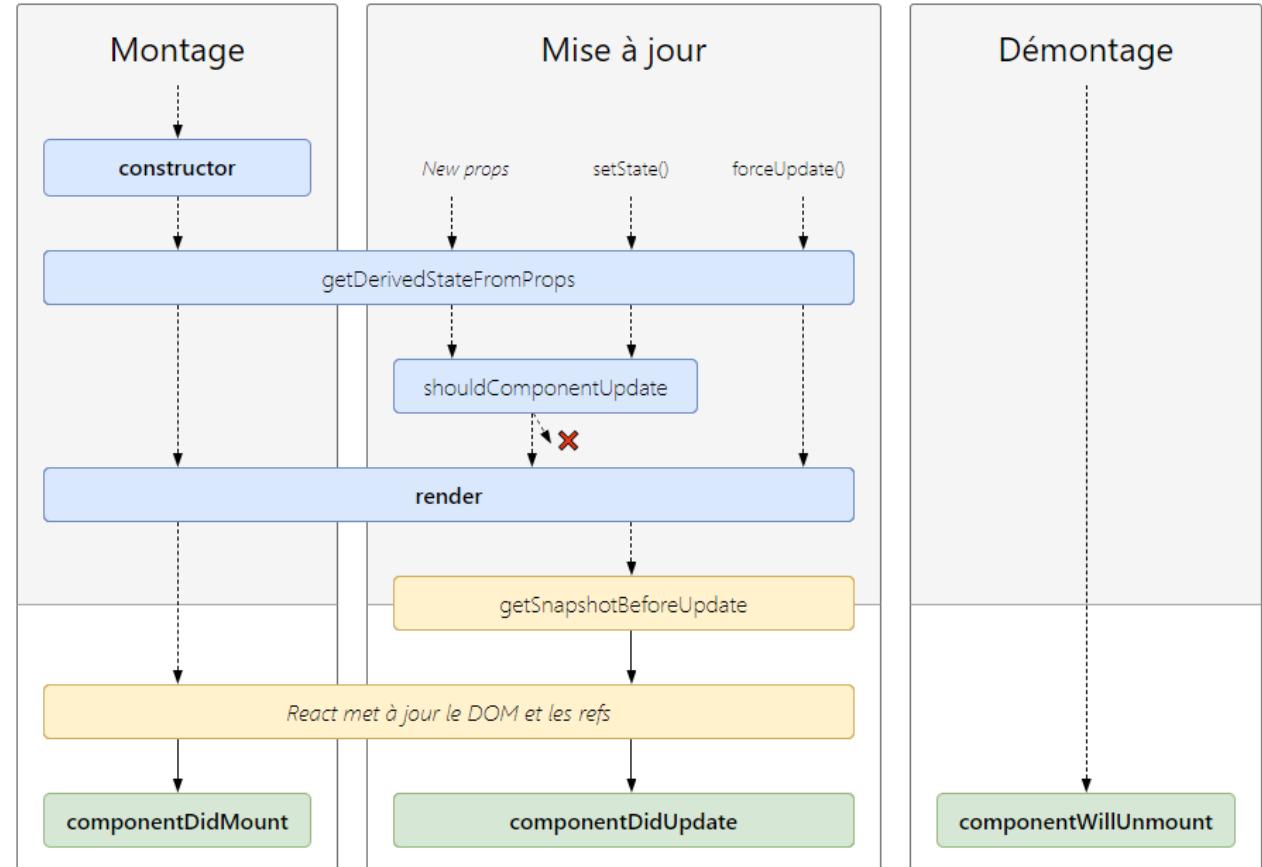
A moins d'être sur d'avoir  
à le faire et que c'est  
**vraiment la seule et  
unique solution possible**



# Le cycle de vie

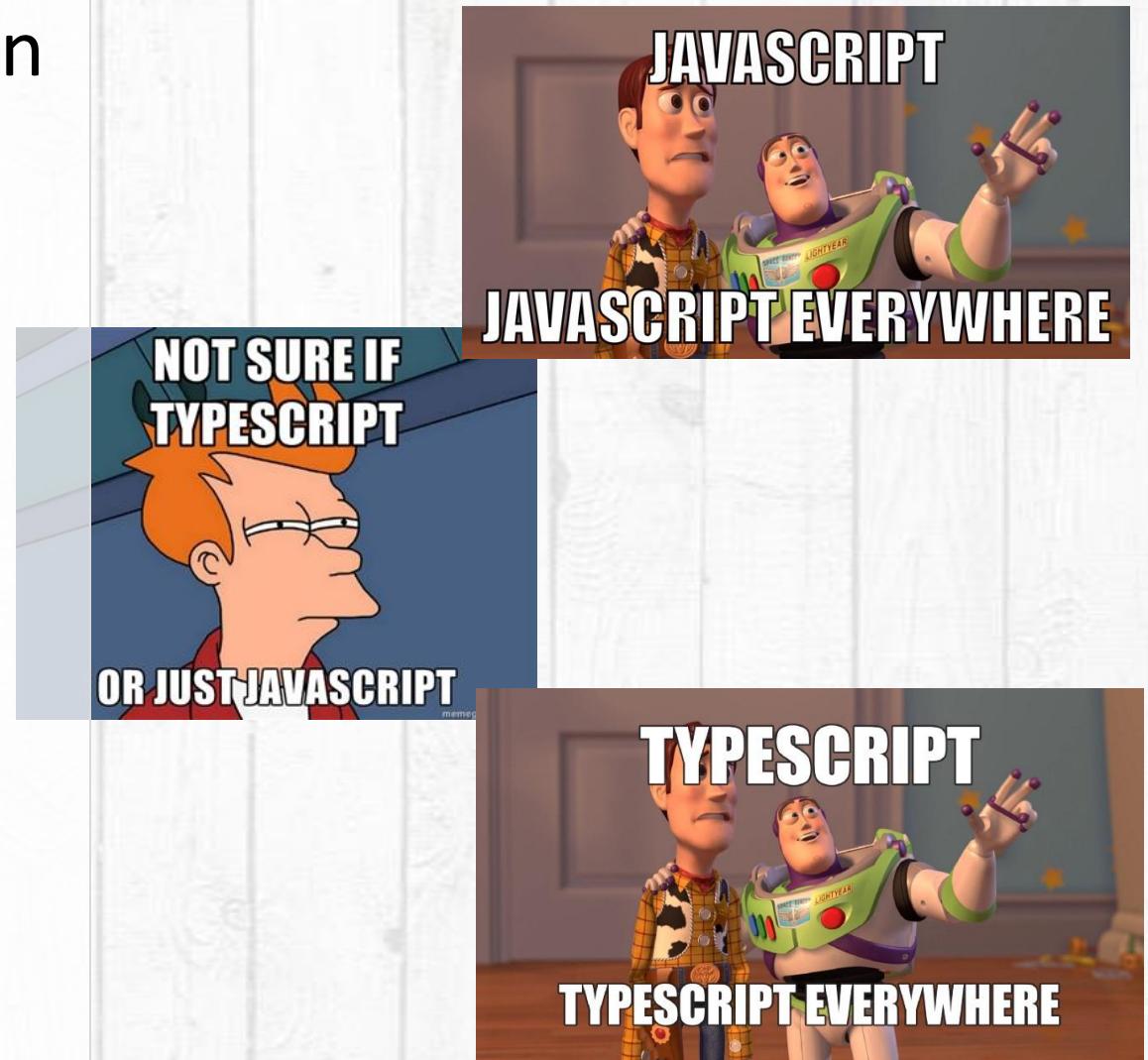
- Tout au long de la vie du component des fonctions vont être exécutées en fonction de l'état et de son contenu

- Doc :  
<https://fr.reactjs.org/docs/react-component.html#the-component-lifecycle>



# Tout le reste n'est que JS

- React ne permet que la confection rapide de l'app
  - Assembler avec des component
- Qui s'occupera du reste ?
  - l'éco-système de react avec les nombreuses librairies tierces
  - Avec des appels de fonctions js native



# Les formulaires

- Le binding :
  - Par défaut seul le ***one way binding*** est présent grâce à :
    - Affiche la valeur de l'état dans l'input
    - `<input value={this.state.value}/>`
  - Méthode de ***two way binding*** :

Chaque champs possède leur propre déclencheur d'événement



- onChange pour effectuer la modification du state/store
  - `<input value={this.state.value} onchange={this.evtfunct}/>`
  - La fonction recevra un paramètre d'entrée avec l'event associé



# Tp faire un composant formulaire

- Faites saisir l'utilisateur :
  - Faites un Button personnalisé
  - Faite un Slider qui affiche on/off
  - Un formulaire avec plusieurs champs
  - Retranscrivez en texte la valeur de l'état
  - Faite un formulaire d'ajout d'image grâce aux liens url de l'image



# Tout le reste n'est que JS

- React ne permet que la confection rapide de l'app
  - Assembler avec des component
- Qui s'occupera du **rest** ?
  - Les appels http
  - Le cycle de vie
    - Le composant est déjà monté
    - **componentDidMount**
      - Initial fetch for a component and son
  - await / async (es7/ts)
    - **async componentDidMount**

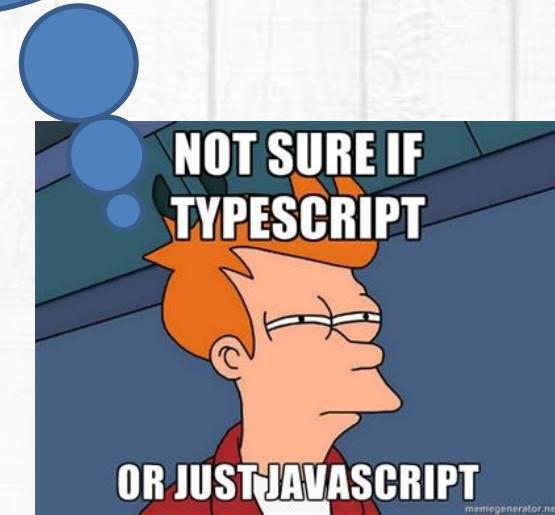


- Finir les fonctions de l'objet CRUD
  - Privilégier la méthode **fetch** aux **xhr**
- Implémenter objet crud dans un component de liste
- De même pour un élément unique de la liste
  - Traiter
  - afficher



# Tout le reste n'est que JS

- Comment faire les appelles réseaux?
  - API fetch
  - Xhr
- Comment parler en composants?
  - Les props
    - Envois de fonction de « *callback* »
  - Approche HighOrderComponent
    - On parle de wrapper de component
    - Un composant gerant un enfant, connu ou passé en children, et de datas,
      - » en régulant le partage
    - fils<-> parent<->app(niveau supérieur)
  - Approche store, redux, mobx, ...



# React-router-dom

- Le GPS qui construit le chemin pour monter l'app
  - En fonction d'une simple adresse
- Le routeur js pour react
  - React-router-dom
  - Associe une URL à
    - des valeurs étatique
    - une représentation de l'état
- Non livré avec CRA
  - Installation via
    - yarn : *yarn add react-router-dom*
    - Npm :*npm install --save react-router-dom*
  - Imports :
    - **Import {Link,Route,Switch,BrowserRouter as Router} from 'react-router-dom'**



# React-router-dom

- Concept R.R.D
  - Notion de routeur
    - » BrowserRouter,
    - » memoryRouter
  - Component Router
    - » Elle englobe tout l'appli concernée par le routage
    - » <Router/>
  - Component Route
    - » Défini le contenu à exécuter selon un chemin (strict ou non)
    - » <Route path="/">
  - Component Link
    - » Equiv <a href=" " > sur des chemins virtuels(pas lié à une arborescence de fichier)

- Exemple de link:

```
<Router>
  <ul>
    <li>
      <Link to="/">Home</Link>
    </li>
    <li>
      <Link to="/about">About</Link>
    </li>
  </ul>
  <Route exact to="/"><Home/></Route>
<Router>
```



# React-router-dom

- Si l'exécution est conditionnelle avec plusieurs issues possibles pour un bloc XML

- Balise Switch

- » <Switch><Route+/></Switch>

- » Une seule route sera exécuter

```
<Switch>
  <Route exact path="/">
    <Home />
  </Route>
  <Route exact path="/meme">
    <Meme />
  </Route>
</Switch>
```

- **Attention** à l'ordre de déclaration des routes.
  - La 1ere à être vrai sera alors la seul exécuter
  - Privilégier la déclaration en 1<sup>er</sup> des routes ayant les profondeurs les plus importante d'url



# Routes & paramètres

- En plus de lié des ensembles de ressources présélectionnées,
- Il est possible de fournir des paramètres dans l'url pour rendre la construction encore
  - plus dynamique
  - Mieux ciblé
  - Exemple : /ressource/:id /ressource/:id?
- HOC & Injections dans les props des datas du router, *withRouter* :  
<https://reacttraining.com/react-router/web/api/withRouter>

- Exemple de param

```
const TestCmp = () => (
  <Router>
    <Route path="/memes/:id" >
      <Child></Child>
    </Route>
    <Route path="/memes" >
      <Home></Home>
    </Route>
  </Router>
);
```

```
const Child = () => {
  let { id } = useParams()
  return (
    <div>
      {id}
    </div>
  );
}
```

# ECOSYSTEME



- Permet une implémentation facile de bootstrap
  - Fonctionne sur les versions
    - 4 (par default)
    - 3.4, ....
- Permet d'avoir des objets de bootstrap sous forme de composants
- Les glyphicons pour bootstrap 3\*\*attention
- Font awesome pour bootstrap 4  
<https://github.com/FortAwesome/react-fontawesome>



- Gestion de map
- Implem. de mapbox-gl pour angular
- Concurrent de google maps api
- <http://visgl.github.io/react-map-gl/>



- Fenêtre modale simple
- Commande
  - npm install react-modal --save
- <https://www.npmjs.com/package/react-modal>

```
<Modal contentLabel="Example Modal">
  <h1>Mon modal</h1>
</Modal>
```



# Redux

- Redux manage un store pour stocker un état pour une app ou un groupe de composants
  - Il nécessite une fonction qui dispatch et exécute les taches possible sur le store
    - Prendre produit, retour produit , payer produit, ...
    - L'ensemble agit sur l'état du magasin
  - Les taches sont affectées a un reducer
    - Son rôle est de
      - » modifier l'état en **un nouvel état** issue du précédent
      - » le **rendre l'état** ou une valeur de l'état
    - Il **agit** en fonction d'**actions** qui lui sont soumis
- React-redux est une intégration plus simplifier des stores dans react



# redux

- Pour gérer une store

- Un reducer

- Reçoit :
    - des actions
    - L'ancien state
    - avec un state initial (valeur par def.)
    - \*la nouvelle valeur à manager dans le state

- Retourne un nouvel état assemblé issue de l'ancien

- Un store

- Le store sera l'interface des composants avec le reducer
    - Il permet d'abonner des fonction aux changements du state du magasin
    - Il **dispatch** les appels au reducer avec le renvoie de l'ancien state qu'il manage
    - » Seul l'action sera à fournir

- Création du store & affectation du reducer

- Abonnement
    - Usage du store

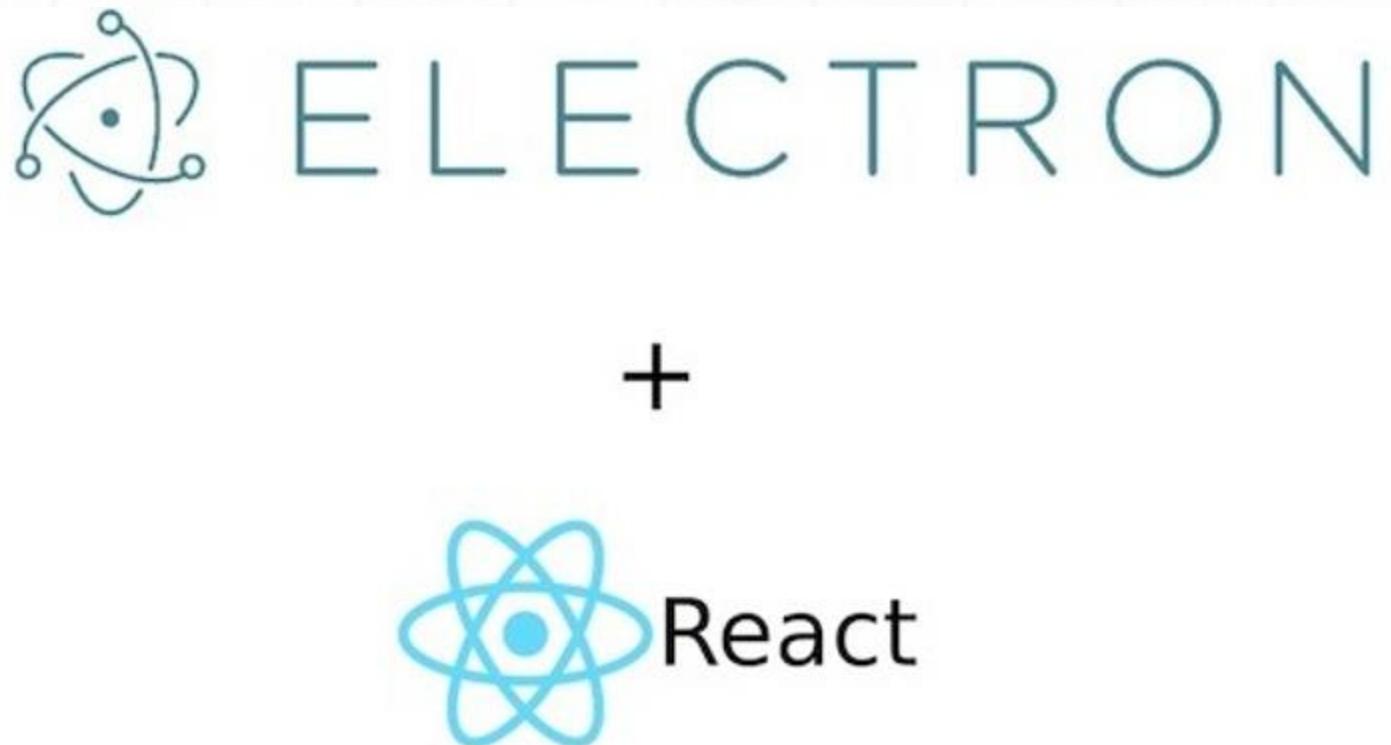
- Exemple

```
function counter(state = 0, action) {  
  switch (action.type) {  
    case 'INCREM':  
      return state + 1  
    case 'DECREM':  
      return state - 1  
    default:  
      return state  
  }  
}  
}  
}
```

```
let store = createStore(counter)  
  
store.subscribe(() =>  
  console.log(store.getState()))
```

```
store.dispatch({ type: 'INCREM' })  
store.dispatch({ type: 'DECREM' })
```

- Electron et react
  - Encapsulation html5/CSS/JS sous forme d'appli « native »
  - Web browser encapsuler



# TOOLS



# Pour les fan du angular/cli

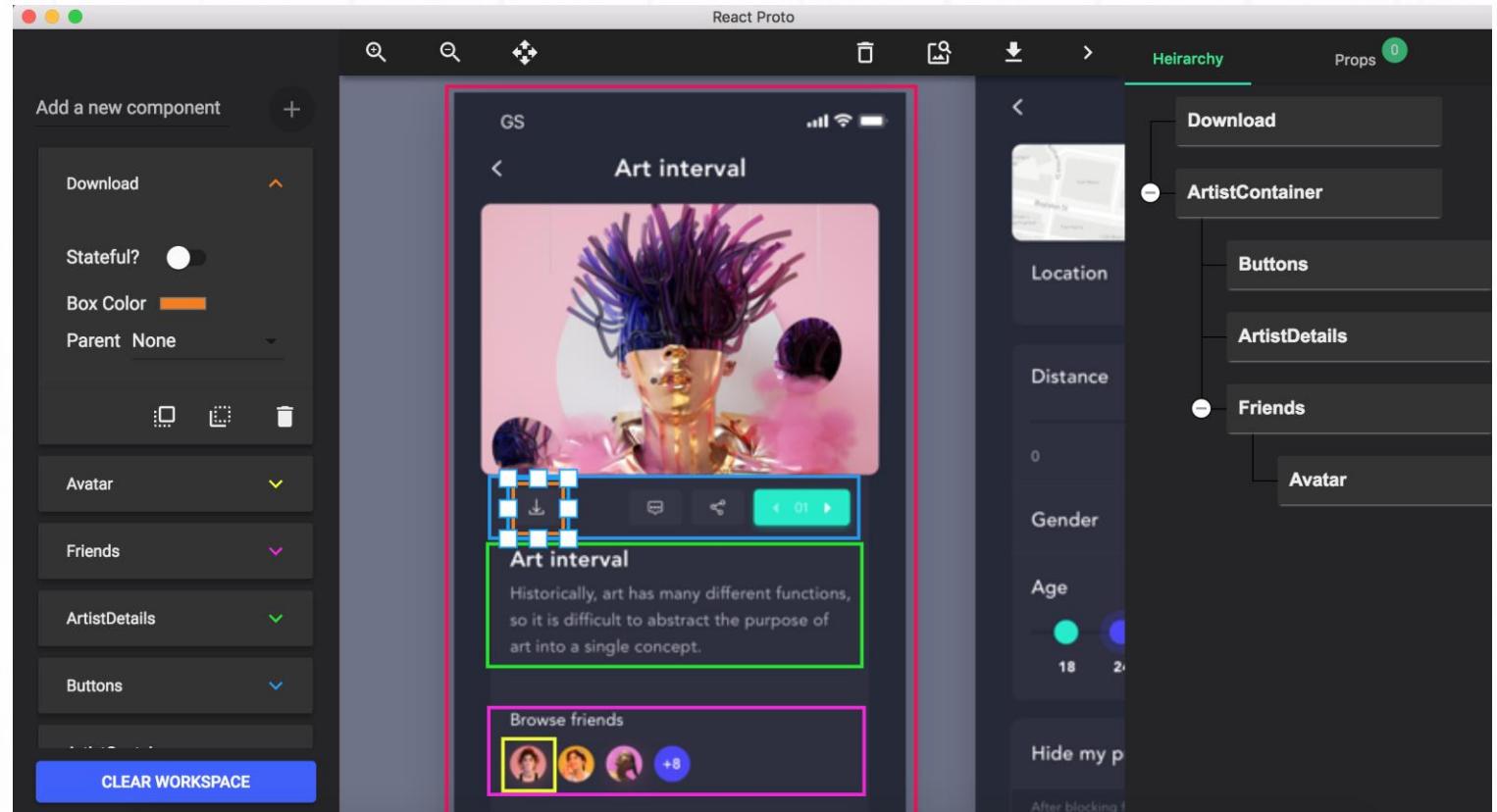
- Generate-react-cli :
  - Un module node.js permet de générer les components
  - Il est configurable
    - Grace à : generate-react-cli.json
    - Fichier généré lors du 1<sup>er</sup> appel
  - Pour configurer sur le projet
    - npx generate-react-cli
  - Usage de la commande :
    - generate-react component|c [options] <name>



# React-proto

- Construction du squelette des modules

- Drag & drop design
- Génération de jsx



- <https://react-proto.github.io/react-proto/>



# Create-react-library

- Permet la création et la diffusion simple de lot de composants sous forme de module
  - Installable par npm/yarn (si publier à npm)
- Créer la structure pour faire une librairie pour react
  - Fournit aussi une structure pour tester le composant
    - Avec installation automatique du/des composants de la lib

Commande :

```
npx create-react-library
```

- Structure générée :

```
~/dev/temp $ cd my-dope-component/
master* in ~/dev/temp/my-dope-component$ tree
.
├── README.md
├── example/
│   ├── README.md
│   └── package.json
└── public/
    ├── index.html
    └── manifest.json
src/
├── App.js
├── index.css
└── index.js
yarn.lock
package.json
rollup.config.js
src/
├── index.js
└── styles.css
test.js

4 directories, 14 files
```

# TP1 : MEME generator

medium.com/free-code-camp/react-for-beginners-building-a-meme-maker-with-react-7164d3d3e55f

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

React for Beginner's guide

Avanthika Meenakshi [Follow](#)  
Dec 23, 2018 - 5 min read

freeCodeCamp.org  
This is no longer updated.  
[Follow](#)

1K

If you're in the middle of learning React, you have probably already been through lots of tutorials on how to build a to-do list. At some point, you'll look for alternate ideas to try and learn but you'll keep bumping into different versions of the default to-do list example.

This alternate idea in this article is for you, the curious ones. The [codebase](#) can be found in my GitHub and it is bootstrapped from [create-react-app](#). I've collected meme-templates from google and other sources. The [Impact](#) font can make any image a meme, so we've got no choice but to add it.

Make-a-Meme



Top Text  
BEST CODE IS THE CODE

Bottom Text  
THAT'S NEVER WRITTEN!

Download Meme!

uploaded\_istock-51...jpg   screencast (1).gif   Annulé   screencast (2).gif   Annulé   screencast (1).gif   Annulé

Tout afficher   Bureau   04:06

- Définition
- Structure
- Formes basiques
- Formes complexes
- Groupes
- Texte, liens, symboles, images
- Filtres
- Textures
- Animations
- Multi formats

# SVG

# Définition

- Scalable Vector Graphics (SVG)
  - Le SVG, langage de graphiques vectoriels, sert à générer des graphiques dynamiquement, à les animer et à les rendre dynamiques.
  - C'est un standard du W3C basé sur XML
  - Utilise les styles CSS pour la mise en forme
  - Contrairement à JPEG ou PNG, il n'est pas trame
  - Son principal concurrent est aujourd'hui Flash
  - Il fonctionne correctement sur les navigateurs récents (plugin MSIE natif MFF)
  - Il existe plusieurs normes SVG, la 1.0 et 1.1 sont les plus utilisées avec une préférence pour cette dernière



- Avantages de SVG
  - Les graphiques peuvent être liés à d'autres ressources via xlink
  - Les SVG sont animés (avec SMIL)
  - Et interactifs car scriptable (on peut faire de véritables interfaces)
  - Les transformations géométriques sont simples (vectoriel) et s'effectuent sans perte : déplacements, agrandissements, fondus, rotations...
  - A un document SVG on peut appliquer plusieurs feuilles de style pour un rendu totalement différent (routes, courbes de niveau, stations, lacs...)
  - SVG intègre le DOM (attention pas le DOMHtml !)

- Inconvénients de SVG

- Les données XML sont verbeuses, les fichiers SVG sont de grande taille mais avec une compression Zlib,
  - » le format est plus léger que ses concurrents
  - » lisible nativement compressé



<https://zlib.net/>

- Structure SVG

- Le fichier porte l'extension ".svg" et sa DTD est :

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"  
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

- La racine des documents SVG est **<svg>**
    - Le xmlns est obligatoire: <http://www.w3.org/2000/svg>
    - **viewBox** définit la fenêtre de contenu
    - **width** et **height** définissent la taille du SVG dans la page

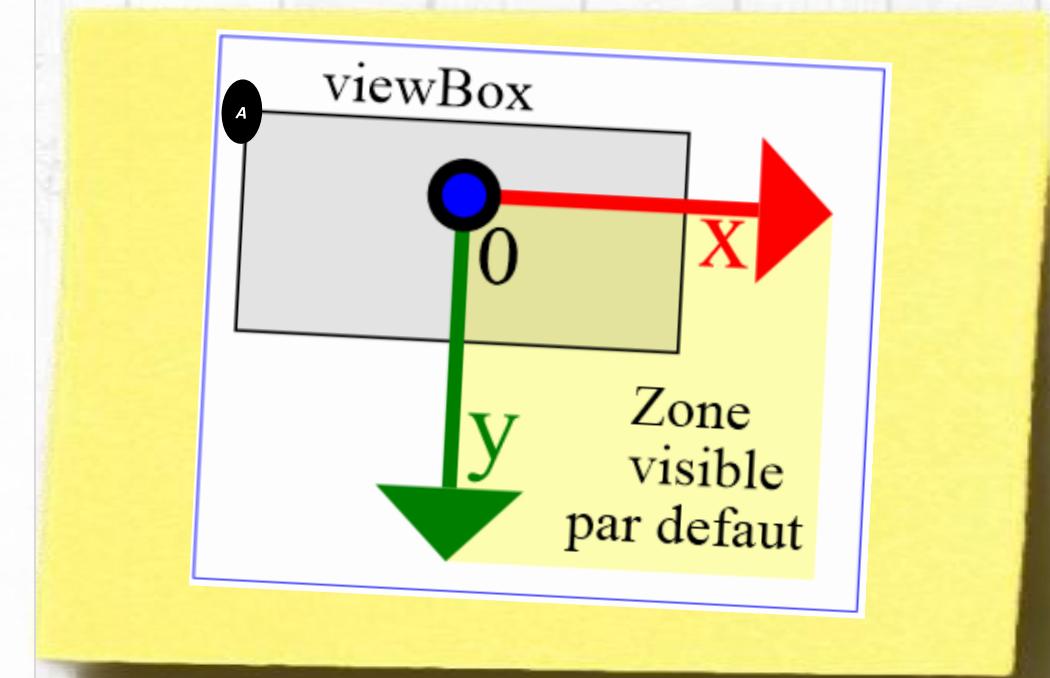
```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"  
viewBox="0 0 500 500">
```

- Certains navigateurs nécessitent de préciser la version="1.1"
  - La section <defs> sert aux définitions(variables, effets, symboles...)

- La balise `<style type="text/css">`
- capacité `@media`
- sélection CSS classique ½ & 3\* si navigateur compatible
  - » `#premierGroup rect{ ... }`
- propriétés css pour déclarer un attribut svg
  - avec le même nom que l'attribut svg
    - exemple :
      - » en css : `fill : red;`
      - » en attribut svg : `fill="red"`

# SVG & Système de coordonnées

- Point de référence et viewBox
- '0' références est le coins supérieur gauche
- Pas d'unité = ordre de grandeur
- viewBox permet d'afficher sur des positions négatives
- **viewBox="Ax Ay width height"**



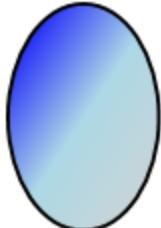
# Formes basiques SVG

- Le dessin svg est assimilable à un jeu de gommettes avec des formes et des tailles variées permettant une fois assemblées de créer des formes complexe
- Les formes de base sont les suivantes :

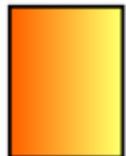
```
<circle r="100"  
       fill="url(#MyGradient2)"  
       stroke="black"  
       stroke-width="5"  
       cx="150" cy="150"/>
```



```
<ellipse rx="100" ry="150"  
        fill="url(#MyGradient2)"  
        stroke="black"  
        cx="150" cy="200"/>
```



```
<rect  
      width="150" height="200"  
      fill="url(#MyGradient)"  
      stroke="BLACK"  
      x="100" y="100"/>
```



```
<line x1="150" y1="50"  
      y2="200" x2="50"  
      stroke="url(#MyGradient)"  
      stroke-width="10" />
```

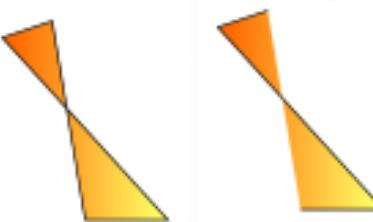


# Formes complexes SVG

- Il existe aussi des balises pour des formes non définies ou pour des besoins particuliers:

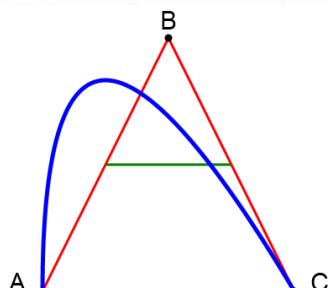
- Le polygone / polylines : forme pleine coordonnées infini

```
<polygon points="170,110  
140,120 240,230 190,230"  
fill="url(#MyGradient)"  
stroke="black"  
stroke-width="1"/>
```



```
<polyline points="170,110  
140,120 240,230 190,230"  
fill="url(#MyGradient)"  
stroke="black"  
stroke-width="1"/>
```

- Le path est un chemin vectoriel aussi appelé trajet ou simplement vecteur. Il permet d'indiquer des mouvements ou des courbures de texte par exemple.



```
<path  
d="M 100 350 q 0 -500 300 0"  
stroke="blue" stroke-width="5"  
fill="none" />
```



# Remplissage et traits

- Remplissage *fill*

- *Couleur css*

- `fill="LIGHTBLUE"`



- *Gradient svg*

- `fill="url(#gradientId)"`



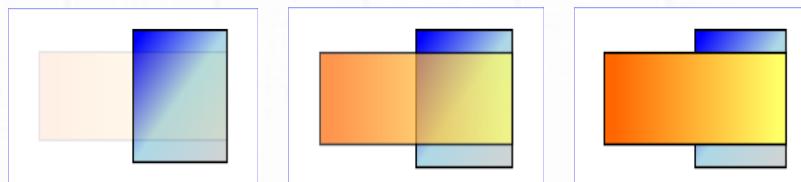
- *Opacité*

- `opacity="value"`

- 0.1

- 0.7

- 1



- Bordures *stroke*

- *Couleur*

- `stroke="BLACK"`



- `stroke="#2E8B57"`



- *Forme*

- `stroke-dasharray="value"`

- "40,10"



- "30,5,10,20"



- *Opacité*

- `stroke-opacity="value"`



- ≈1



- ≈0

- *Epaisseur*

- `stroke-width="value"`



120



- Le groupe **<g>**
  - On peut regrouper certains éléments à l'aide de la balise **<g>** (group) pour des raisons de facilité
    - de déplacement
    - d'application d'effets
    - de sélection de groupe

拇指图标 Mettre un **id** et/ou une **class** pour l'identifier au css

# Texte, liens, symboles, images

- Textes
  - On peut utiliser la balise **<text>** pour insérer du texte
- Liens
  - On peut créer des liens à l'aide de xlink (pas seulement des liens )
    - **<a xlink:href="..."** avec `<svg xmlns:xlink="http://www.w3.org/1999/xlink"...`
- Symboles
  - Pour gagner en temps (load) et poids des fichiers on peut utiliser des symboles
  - On définit un symbole dans **<defs>** avec **<symbol id="mysymbol">** et **<g>**
  - On l'utilise avec **<use xlink:href="mysymbol" x="..." y="..." />**
- Images
  - On peut insérer des images avec **<image xlink:href="myimage.jpg" ... />**



- Textures
  - Pour remplir les formes avec des motifs évolués, SVG propose les gradients
    - *linearGradient* dégradé linéaire
    - *radialGradient* dégradé radial
  - On les définit dans la partie <defs>
  - On l'applique sur les objets à l'aide de l'attribut **style="fill:url(#mygradient);"**
- Rappel RVB (RGB)
  - RVB = Rouge Vert Bleu (RGB en anglais)
  - Ce sont des spectres d'additions (rouge+vert=jaune et pas vert+jaune=bleu), qui combinées, permettent d'obtenir les couleurs de bases
  - Chaque intensité de couleur va de 0 à 255 (0 = éteint et 255 = allumé au max)

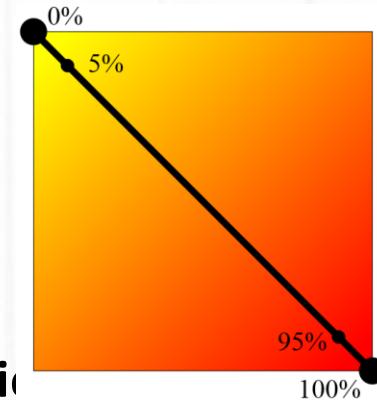
# Gradient SVG (dégradés)

- Dégradés
  - Transition de couleur et/ou d'opacité est possible grâce aux gradient soit linéaire, soit circulaire (dit radiaux)

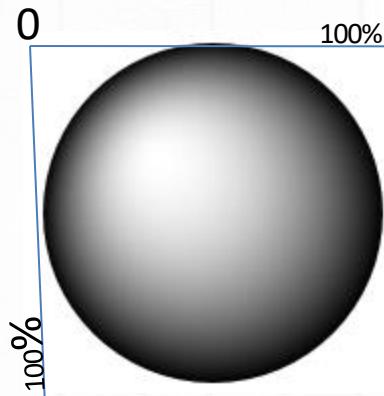
```
<defs>
```

```
  <linearGradient x1="0" x2="100%"  
    y1="0" y2="100%" id="MyGradient">  
    <stop offset="5%" stop-color="yellow"/>  
    <stop offset="95%" stop-color="red"/>  
  </linearGradient>  
</defs>
```

linearGrad



- radialgradient



```
<defs>
```

```
  <radialGradient id="grad1" cx="50%" cy="50%"  
    r="50%" fx="33%" fy="33%">  
    <stop offset="0%" stop-color="white"  
      stop-opacity="0"/>  
    <stop offset="100%" stop-color="darksilver"  
      stop-opacity="1"/>  
  </radialGradient>  
</defs>
```



- Filtres
  - Les filtres sont des effets appliqués à une forme, par exemple un flou, une déviation, une ombre portée... On note, par exemple, les filtres suivants :
    - **feBlend** → mode d'intersection (cf. logiciels de graphisme)
    - **feDiffuseLightening** → lumière diffuse
    - **feFlood** → remplissage
    - **feGaussianBlur** → flou circulaire
    - **feMerge** → fusion
    - **feOffset** → décalage de coordonnées dx, dy de la forme de base
  - On les définit dans la partie **<defs>** avec **<filter id="myfilter"> <feABC />**
  - On l'applique sur les objets à l'aide de l'attribut **style="filter:url(#myfilter);"**



# Animations

- Animations
  - Pour déclencher des effets visuels, on utilise **<animate>** avec les attributs :
    - "**attributeType**" permet de préciser sur quel langage on va jouer (ex: CSS | XML)
    - "**attributeName**" indique quel paramètre on va modifier (ex: **opacity**)
    - "**from**" et "to" indiquent les valeurs de départ et d'arrivée (ex: 1 à 0)
    - "**dur**" indique la durée de l'effet (ex: 3s)
    - "**begin**" pour retarder / synchroniser les animations (ex: 2s)
    - "**repeatCount**" permet de répéter l'animation un nombre de fois précis (ex: 3)  
avec les paramètres de l'exemple on aurait un effet de fondu qui se lancerait 3x après 2s
  - Pour animer une forme on utilise **<animateMotion>** avec les attributs :
    - "**path**" permet de préciser le chemin (vecteur, trajet) à suivre en un temps donné par "dur"
  - Avec **<animateColor>** on peut aussi passer d'une couleur (from) à une autre (to)
  - Et on fait des transformations à l'aide de l'attribut **transform="translate(x,y)"** ou de **<animateTransform>** (ex: **type="rotate | scale | translate | skewX/Y..."**)



- Génération multi format : **XHTML/Fo + SVG en XSLT**
  - On remarque qu'avec les possibilités d'**XSLT** et de **SVG** on peut extraire des données XML et créer **dynamiquement** des **graphiques** SVG
  - Donc **générer des images à partir de données**
  - C'est très utile en particulier pour les **outils statistiques** tels que les outils d'aide à la décision, les agrégations peuvent être représentées graphiquement à la volée
  - De plus, le tout peut être **embarqué** dans une page **xHTML** générée ou **XSL-Fo**, elle aussi **dynamiquement** en **XSL-T**, ceci nécessite quelques précautions, dont :
    - l'utilisation de namespaces pour éviter la confusion des langages utilisés
    - sous MFF, l'usage de certaines extensions (".**xhtml**" au lieu de ".**html**" ou ".**htm**")
    - sous MSIE, le recours à une balise **object** pour forcer l'utilisation de la visualisatrice **SVG** même pour du **SVG** créé à la volée en texte mêlé au **XHTML**