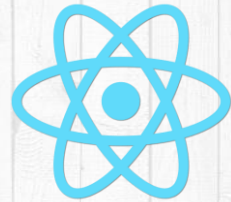




ORSYS
formation

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, ccs, angular, react, ...
- blogger sous Wordpress

Bienvenue chez Orsys

logistique



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



PRÉSENTATION



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 !!!!

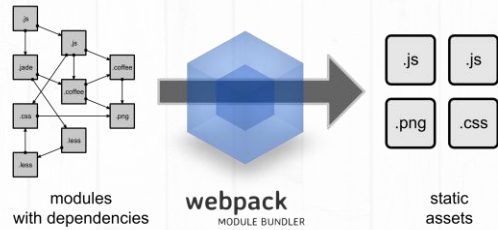
Une librairie

- 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

Web pack : <https://webpack.js.org/>

Babel : <https://babeljs.io/>

Une commande

*option
fortement
recommandée

- React est fournit 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 generation(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](https://github.com/arminbro/generate-react-cli)

Un article traitant de l'outil de generation : <https://dev.to/arminbro/generate-react-cli-1ooh>

CRA

- 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: « »

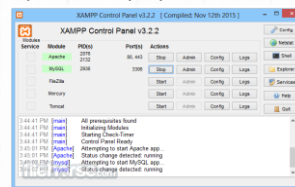


XAMPP

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



- Mdp root mysql: « »



Voir article : <https://www.blogpipers.com/2015/06/lamp-stack-xampp-vs-wamp-vs-mamp/>

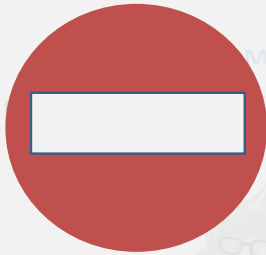
Windows Apache Mysql Php

WampServer



XAMPP

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

Voir article : <https://www.blogpipers.com/2015/06/lamp-stack-xampp-vs-wamp-vs-mamp/>

Choix d'un serveur

- Servir du html en production :

- IIS
- Apache
- Nginx, node, ...

- Dynamisation

- Coté client js
 - IIS, apache, node, ...
- Coté serveur js (code isomorphe)
 - Node

- Scaffolder

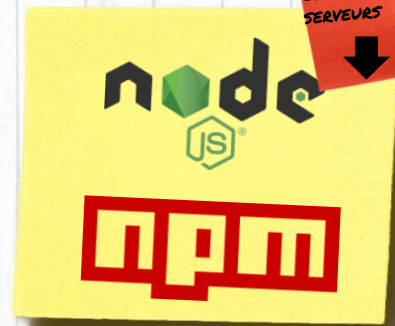
- create-react-app
 - Node.js
- generate-react-cli
 - Node.js,

ROLES

- SERVIR LES FICHIER
- GÉRER LE JS COTÉ SERVEUR *OPTION
- OUTILS DE DEV

Node.js & npm

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

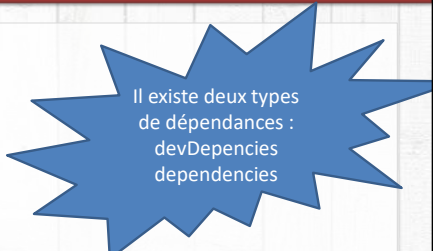


UTILISE DANS LE
COURS POUR LES
SERVEURS

npm

- Commandes npm

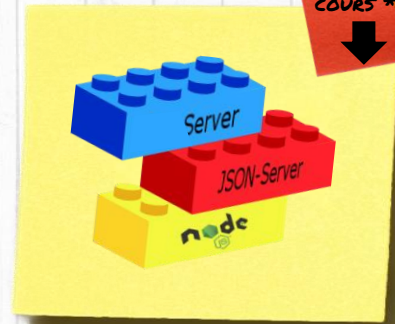
- Initialisation d'un répertoire de projet
 - Créer un fichier minimal package.js
 - » **npm init** -y
 - Pré-configurer node avec le répertoire .git (si .git présent)
 - -y : initialisation « quiet »
- Installation des paquets
 - » **npm install -g packetname** packet2@version
 - -g : installation pour l'utilisateur courant,
 - sans -g installation pour le projet, dans le répertoire node_modules
 - --save permet l'ajout du paquet 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



***ou wamp**

JSON-server tuto : <https://codingthesmartway.com/create-a-rest-api-with-json-server/>

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

VS Code

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

UTILISE
DANS LE
COURS

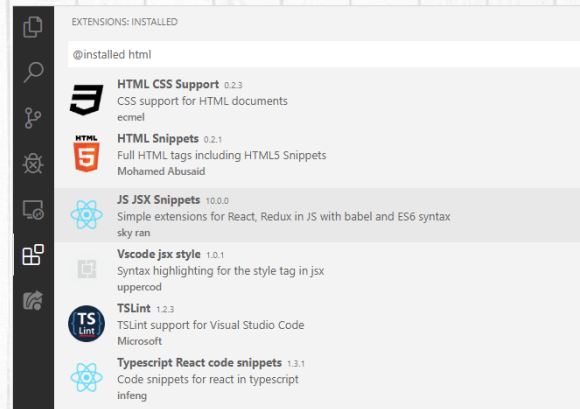


Sublime Text 3

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

Vs code les plugins

- **Css3 support**
 - Ajout de complétion
- **Html support**
 - Ajout de complétion html
- **Js jsx snippet**
 - Snippets
 - esLint
- **Typescript react code snippet**
 - Snippets
 - TsLint



Typescript react snippets

tsrcc→ class component skeleton	scu→ shouldComponentUpdate method
tsrcfull→ class component skeleton with Props, State, and constructor	cwu→ componentWillUpdate method
tsrcjc→ class component skeleton without import and default export lines	cdu→ componentDidUpdate method
tsrpcc→ class purecomponent skeleton	cwum→ componentWillUnmount method
tsrpcjc→ class purecomponent without import and default export lines	gdsfp→ getDerivedStateFromProps method
tsrpc→ pure function component skeleton	gsbu→ getSnapshotBeforeUpdate method
tsdrpfc→ stateless functional component with default export	sst→ this.setState with object as parameter
tsrsc→ stateless functional component	bnd→ binds the this of method inside the constructor
conc→ class default constructor with props and context	met→ simple method
cwm→ componentWillMount method	tscntr→ react redux container skeleton
ren→ render method	imt→ create a import
cdm→ componentDidMount method	
cwrp→ componentWillReceiveProps method	



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
- Atlassian BitBuckets *(Sélectionné pour le cours)
 - Multi-repository
 - Free (max 1Go/repo) ou payed account
 - Officiel uniquement

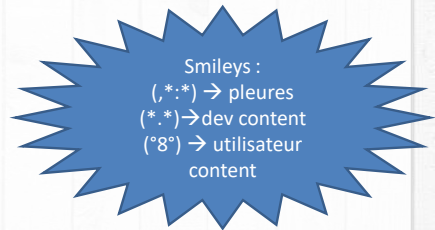


Git - bitbuckets

- 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^{er} 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°)



*`alert()` est bloquant pour l'execution du code js

**`console.log` permet de logger des object complets en gardant l'async

Configuration Node.js

- Node js
 - Moteur de code js coté 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 erreur parfois sourde ou juste des mauvaises pratiques
- L'usage de ' ... ' pour les chaines de caractères dans le js
 - `Var maChaine ='Demat breizh';`
- Tous les blocs possèdent des { ... }
 - `If(...){ ... }`
- On limite l'usage des *eval* ()
- On privilégie l'opérateur === ou !==
 - `a === b`
- On met les constantes en premier pour les comparaisons
 - `if(undefined === maValue)`

Es7 2016 / Es6 2015

- ES 5 est succéder par différentes versions
 - Apportant de nouvelles fonctionnalités
 - Es6 (ou es2015)
 - puis es7
 - ,...
- Les navigateurs supportes a coup sur l'es5 pas forcément les version supérieurs
 - Babylon.js convertie le code js dans différentes versions cibles

Nouveautés syntaxiques es6 :

- L'usage du mot clef class pour la définition d'objets
 - Mot clef private, public protected
 - 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
 - Une promesse d'exécution grâce à `then(()=>{})`
 - La nouvelle portée de scope : le module

Nouveautés syntaxiques es6 :

- Les template strings

- Construction de chaîne finale

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

```
${chaines[0]}${varI}.
```

- La chaîne est définie avec l'accent grave et non des apostrophes

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

- Peut effectuer des choix, ternaires, ...

Doc mdn des templates string : appelé des « *littérales et gabarits* » :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Litt%C3%A9raux_gabarits

Syntaxe es6

- Spread operator

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

```
let tab1=[elem2,elem3];
```

```
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 avait une course de chevaux qui allait démarrer
 - Il a donc fait un pari sur un cheval
 - Il attend devant la course qui débute
 - Son cheval est placé
 - Il remporte sa mise

Ici une notion d'asynchronicité aurait pu être mise en place

Jp aurait pu attendre boire un petit café tranquillement ou tout simplement revenir demain

Peut être que toute sa vie reposait sur ce pari

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



Ici une notion d' asynchronicité aurais pu etre mis en place

Jp aurais pu attendre boire un petit café tranquillement ou tout simplement revenir demain

Peut etre que tout sa vie reposais sur ce parie, il a donc tout perdu

Tout le reste de sa vie dépendais de ce paris il a donc attendu car il ne pouvait rien faire d'autre tant qu'il ne savais pas le déroulement de la suite des choses

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

- Enregistrer 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 jetterais son ticket

- Vue qu'il étai 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 jeu à gratter n'avait aucune notion d'async
- La course de chevaux n'a pas été utilisée sous forme d'async
- Notion de promesse es6 et **await** fonction 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 finie j'exécute une tâche
 - En sachant à l'avance quelle prendra du temps et que j'aurai bien d'autres tâches à effectuer d'ici le résultat de la fonction appelée.
 - Je prévois directement, au moment de l'appel de la fonction, le **retour d'une promesse**.
 - Promesse à laquelle je peux définir quoi faire quand l'exécution sera effectuée
 - 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

Fonction encore expérimentale

Les wrapper High order peuvent être définis par une simple fonction.

Le décorateur apporte surtout de la lisibilité et de la maintenabilité du code en le rendant plus simple à lire et à écrire.

Il a pour seul rôle de substituer un `@quelqueChose()` à une fonction pour rendre plus abstrait et donc plus simple certains fonctionnements

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
 - Cocreer 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 déclarée.

- (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
```

Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

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

let names: `Array<string>`;

- Le typage des *array* possède plusieurs syntaxes:

- Il possède aussi les syntaxes suivante en plus de la syntaxe générique

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

- Typage pour de fonction
function hello(

```
  nom: string,  
  age: number = 25,  
  isMale?: boolean  
): string  
{return `Demat ${nom} ${age}` ;}
```

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

Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

Class

- Les classes
 - permettent la 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 appartiennent qu'à lui
 - Notion de **private**, **protected**
 - Il peut avoir des champs/fonctions disponibles pour interagir avec lui
 - Notion de **public**

Génèrera une fonction pour l'es5 représentant l'objet transformé **sans class**

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 prénommé Jean-Pierre

- Identifions une class machine à café

- Des fonctions d'interaction **public**
 - Elle à une fonction marche / arrêt
 - 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!!!



On parle ici de l'extension d'une class

qui reprend tout ce que fais un objet pour faire un nouvelle objet a partir du premier.

Il peut redéfinir des fonctions ou méthodes existante chez le parent et en apporter d'autres.

Ici notre cafetier possède les memes fonctions : faire le café par exemple

Mais le fonctionnement n'est pas profilement le meme.

Le café cette fois , toujours présent mais un nouveau conteneur doit etre insérer dans la machine , ... plus le café directement

Class machine à café

- Machine à café à dosettes
 - Ses champs privés
 - Ceux du **parent** étant eux aussi **private** il sont **inaccessibles**
 - » Notion d'héritage
 - » Notion de **protected**
 - Des champs / fonctions public, protected
 - Permettant l'héritage aux futures extensions
 - **Redéfinition** de la manière de faire le café
 - » Notions d'**override**
 - Des champs / fonction privés qui lui sont totalement personnels
 - Un constructeur
 - Appel du constructeur parent
 - » Notion de **super()**
 - Des fonctions internes protégées pour permettre aux futures générations de pleinement hériter de leurs aînés

```
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 : mousser à lait
- Il a aussi identifié 2 sous modèles génériques pour faire du café
 - Les machines classiques
 - Les machines a espresso



On parle d'abstraction.

Dans l'abstraction on définit (ou uniquement le besoin de de déclarations), le contenu commun a implémenter chez l'enfant

L'abstraction est plus a voir comme la définition d'un *Object* générique plutôt *sous forme de besoins conceptuel*

Contrairement à **extends**, **implements** se limite à forcer l' existence de fonctions ou champs pour chaque implem.

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

- Jean-Pierre est devenu un expert du café
 - Il a essayé une 20aine de cafés différents.
- Il à finit par déduire que
 - Le café est toujours caractérisé de la même manière mais nécessite pas un besoin d'objet défini
Seul le fait qu'il contienne tout me suffit
 - Il utilise donc une interface qui définit juste le contenu décrit
 - Arôme,
 - Provenance,
 - ... mais **aucune fonctionnalité** à 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 différents.
- 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
 - Arôme,
 - » corsé, doux
 - Provenance,
 - » Pérou, Mexique, ...
 - ... mais **aucune fonctionnalité** à proprement parler
- Il formalise et catégorise pour une valeur



On parle ici d'énumération,

Chaque ensemble est définissable comme un type à part en tirant toujours un des champs disponibles pour l'énumération

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';
```

Renommer un import avec as

Import { exportedVar as myNewVarName } from './fichier'

Il existe d'autres patterns pour la construction de module

Ces autres patterns sont pris en charge par web pack

ex: pattern amd et la fonction **require**('...');

tsc

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

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

Commande pour initialiser le fichier tsconfig.json:

tsc -init

Générer dans le tsconfig

```
"target": "es5",                /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', or 'ESNEXT'. */
"module": "commonjs",          /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020', or 'ESNext'. */
```

Tsconfig.json

• EXEMPLE DE FICHIER TSConfig.JSON GENERER PAR TSC -INIT

```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5", /* Enable incremental compilation */
    "module": "commonjs", /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES6', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', 'ES2021', 'ES2022', 'ESNEXT'. */
    "lib": [], /* Specify library files to be included in the compilation. */
    "allowJs": true, /* Allow javascript files to be compiled. */
    "checkJs": true, /* Report errors in .js files. */
    "outFile": "", /* Concatenate and emit output to single file. */
    "rootDir": "", /* Redirect output structure to the directory. */
    "removeComments": true, /* Do not emit comments to output. */
    "noEmit": true, /* Do not emit outputs. */
    "isolatedModules": true, /* Transpile each file as a separate module (disables many compiler options) */
    /* 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 metadata for 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

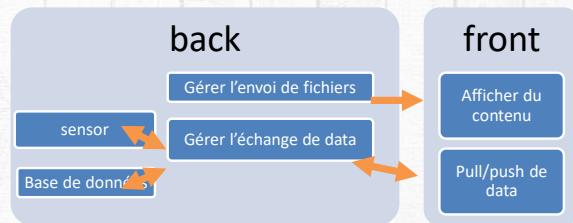


Qui fais quoi ?

- Qui à besoin de quoi ?
 - Le client, Un navigateur
 - Charge le site et met en action le JS
 - Il gère grâce au js les interactions IHM
 - Charger une version du site html/css/js
 - Et les librairies js associées
 - Il gère 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 gère des échanges internes pour l'accès au sgbd, 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



Les services Web

- 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)

Précisions : Les services web sont le résultat de la mise à disposition d'un service développé par A sur Internet. Il est ainsi possible de les utiliser pour le développement d'un site. Ils permettent par exemple d'inclure un encart avec la météo sur sa page d'accueil sans avoir à le faire soi-même. Il faudra simplement intégrer ce service à sa page. Il est possible d'utiliser les services Web comme des briques de son site.

REST (*Representational State Transfer*) : Est une architecture logicielle, utilisée sur le Web pour décharger le serveur qui n'a pas besoin de gérer l'état des transactions (c'est le client qui sauvegarde toutes les variables utiles aux traitements métiers). Dans l'imaginaire des adeptes du Web agile, REST est une alternative à SOAP plus efficace, en fait les deux techniques ne sont pas incompatibles. On peut faire des échanges SOAP selon une architecture REST.

RPC (*Remote Procedure Call*) : c'est un appel de procédure distante = qui s'exécute dans un autre espace d'adresse (ordinateur local ou distant), c'est à dire qu'on demande (call) à un ordinateur distant (Remote) de faire une opération (procédure) pour nous et de nous renvoyer éventuellement son résultat (« tout c'est bien passé ». données en retour...).

Un bistrot sur le web?

- Le bar symbolise:
 - Réserve
 - les données, sgbd, 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 servi en fonction du contexte
 - 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



REST

- **REpresentational State Tranfert**
 - 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)

REST

- Accès CRUD

- **C**reate
- **R**ead
- **U**ppdate
- **D**eleter

- Accès à une ressource par HTTP

- Méthodes HTTP
 - POST (create)
 - GET (read)
 - PUT (update)
 - DELETE (delete)

- Capacités

- Filtrage
 - champs
 - id
- Positionnement
- Relationnel *

Echanges
JSON

- 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)

Il existe **différentes** types de **bistrot** REST/SOAP/RPC

La **finalité** est la **même**

L'**ambiance** et le **process** pour commander sont **différents**

La **manière** dont c'est servi peut **changer**



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

Do I tell him
fetch() and then()?



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 le JSON
- En js Historiquement les appels 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

Update compatibility data on GitHub

	Desktop					Mobile				
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android	Chrome pour Android	Firefox pour Android	Opera pour Android
Fetch	42	14	39	Non	27	10.1	42	42	39	27
Support for blob: and data:	43	79	?	Non	?	?	43	48	?	?
referrerPolicy	52	79	52	Non	38	11.1	52	52	52	41
signal	86	16	57	Non	53		66	66	57	47
Streaming response body	43	14	Out	Non	28	10.1	43	43	Non	Non
									10.3	4.0

Examinons fetch

- Method GET par défaut
 - 1^{er} 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 fourni en 2eme arg de fetch
 - `fetch('https://www.reddit.com/r/javascript/.json', {
 method: 'post',
 body: JSON.stringify(opts)
}).then(...).then(...)`

• EXEMPLE GET :

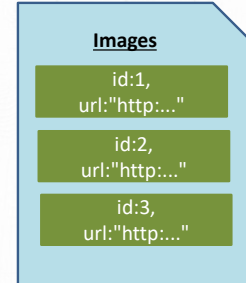
```
const myImage =  
document.querySelector('img');  
fetch('flowers.jpg')  
  .then(function(response) {  
    return response.blob();  
  })  
  .then(function(myBlob) {  
    const objectURL =  
      URL.createObjectURL(myBlob);  
    myImage.src = objectURL;  
  });
```

Utiliser fetch :

https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

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



Fichier json:

```
{
  "images": [
    {
      "id": 1,
      "url": "img/plonge.jpg"
    },
    {
      "id": 2,
      "url": "img/kid1.jpg"
    },
    {
      "id": 3,
      "url": "img/futurama1.jpg"
    },
    {
      "id": 4,
      "url": "img/fry1.jpg"
    }
  ]
}
```

```

        {
            "id": 5,
            "url": "url/farnsworth.jpg"
        },
        {
            "id": 6,
            "url": "img/trololo.jpg"
        }
    ],
    "memes": [
        {
            "id": 1,
            "image_id": 1,
            "text": [
                {
                    "x": 0,
                    "y": 0,
                    "value":
"dev react"
                }
            ]
        }
    ]
}

```

Y' A QUOI DEDANS ?



CONCEPT REACT



CRA

- 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
 - Scaffolding 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, modele 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 creer

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 mimifier du code source
 - **Version de production**
- Test
 - Exécute les commandes de test de l'appli

Arborescence

- folders

- node_modules

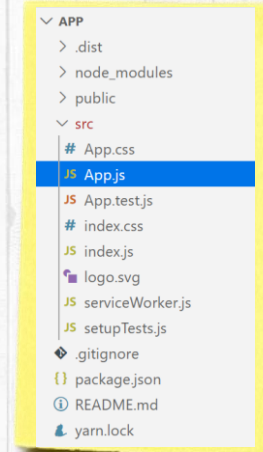
- Modules nodes (nécessaire a node js, create react app et autres lib, ...)

- public

- Static assets
 - Contenu public constant , image , icones, ...

- src

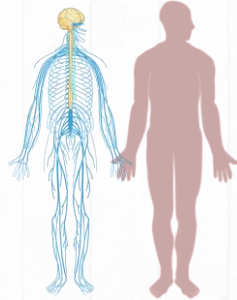
- Code js / css / scss / jsx / ts/tsx...



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 / .ts* / .tsx*
 - » Le code react du composant
- .test.js
 - » Description d'un composant
 - » Un contrôle que le composant se crée correctement



Jsx/ts

css/scss/...

• STRUCTURE GENEREE PAR CRA POUR LE REPERTOIRE SRC

```
src
├── App.css
├── App.jsx
├── App.test.js
├── index.css
└── index.js
```

L'extension .jsx / .tsx permet une meilleure prise en charge du code sous vs mais est facultatif un simple .js / .ts il sera traiter comme du potentiel jsx/tsx
Image de wikipedia : https://fr.wikipedia.org/wiki/Moelle_spinale

Analysons le répertoire src

- Structurer correctement votre app

- Notion de découpage de component

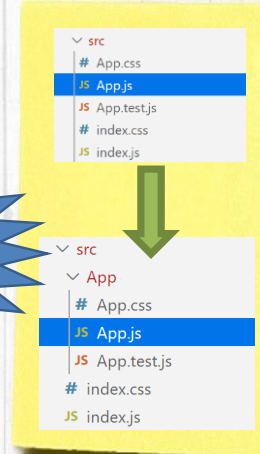
- Feature component
 - UI component
 - Application component

- 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



L'extension .jsx / .tsx permet une meilleure prise en charge du code sous vs mais est facultatif un simple .js / .ts il sera traiter comme du potentiel jsx/tsx

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
défini



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

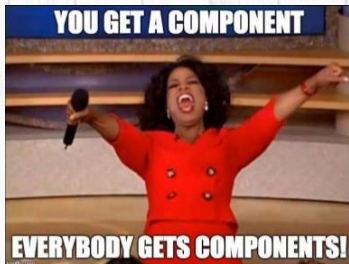
- Reçoit des props
 - Dans le constructeur (*constructor*) de la class
- Avec ou sans états interne
 - Statefull/stateless
 - `this.setState()`, `this.state`
- Possède une fonction **render()** qui retourne le contenu du composant assemblé avec props & état(*state*)
- Extension de class
- Test plus lourd 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" >  
      {/* parcours d'array de logos */}  
      {  
        props.logos.map(  
          e=>{<img src={logo}/>})  
        }  
      </div>  
    );  
  }  
  export default App;
```

Jsx subset

- 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"/>);  
}
```



Nom des composants
avec Majuscules
React.Component !=
html

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

Il **reçoit ses pops 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

PropTypes

- Contrôle du type de contenu pour les props :

- propTypes
- 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 propTypes 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



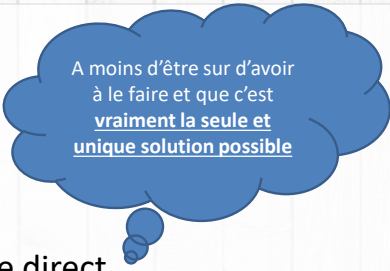
Class et state

- Le state doit être initialiser dès la création

- Dans le constructeur :

📄 **Cmp.jsx**

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



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

- 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

<https://fr.reactjs.org/docs/react-component.html#shouldcomponentupdate>

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>

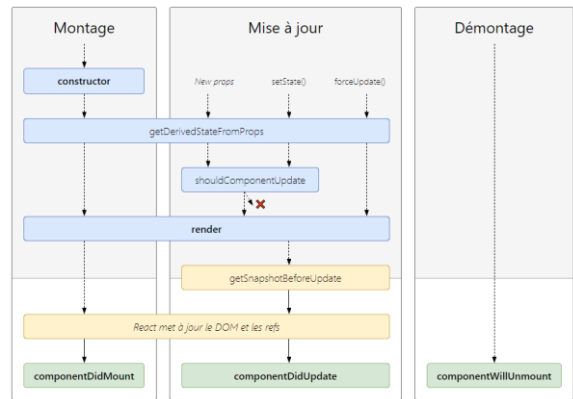
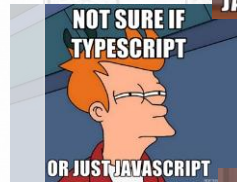


Diagram : <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

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'événement associé

React.PropTypes a été déplacé dans un autre module depuis React v15.5. Merci de plutôt utiliser [le module prop-types](#).

Tp faire un composant formulaire

- Faites saisir l'utilisateur :
 - Faites un Button personnalisé
 - Faites un Slider qui affiche on/off
 - Un formulaire avec plusieurs champs
 - Retranscrivez en texte la valeur de l'état
 - Faites 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

Do I tell him
fetch() and then()?



Tp fetch

- 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 appels 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`, ...

```
<App>  
  <HighOrder>  
    <component/>  
  </HighOrder>  
</App>
```



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
 - » Définit le contenu à exécuter selon un chemin (strict ou non)
 - » <Route path="/">
- Component Link
 - » Equiv 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>
```

<https://reacttraining.com/react-router/web/api/HashRouter>

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 1^{ÈRE} À ÊTRE VRAI SERA ALORS LA SEUL EXÉCUTER

- PRIVILÉGER LA DÉCLARATION EN 1^{ÈRE} DES ROUTES AYANT LES PROFONDEURS LES PLUS IMPORTANTE D'URL

Routes & paramètres

- En plus de lier 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>  
  );  
}
```

<https://reacttraining.com/react-router/web/example/url-params>

ECOSYSTEME



React-bootstrap

- Permet une implémentation facile de bootstrap
 - Fonctionne sur les version
 - 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>

React-map-gl

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

react-modal

- 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 a fournir
 - Création du store & affectation du reducer
 - Abonnement
 - Usage du store

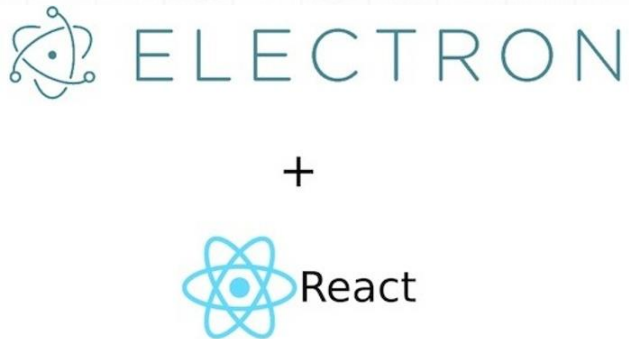
- **EXEMPLE**

```
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' })
```

*facultatif

electron

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



<https://www.codementor.io/@randyfindley/how-to-build-an-electron-app-using-create-react-app-and-electron-builder-ss1k0sfer>

<https://dev.to/jsmanifest/create-your-first-react-desktop-application-in-electron-with-hot-reload-4jj5>

TOOLS



Pour les fan du angular/cli

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

1ere intialisation :

? Does this project use TypeScript? No

? Does this project use CSS modules? Yes

? Does this project use a CSS Preprocessor? css

? What testing library does your project use? Testing Library

? Set the default path directory to where your components will be generated in?
src/App/components

? Would you like to create a corresponding stylesheet file with each component you generate? Yes

? Would you like to create a corresponding test file with each component you generate? Yes

? Would you like to create a corresponding story with each component you generate? Yes

? Would you like to create a corresponding lazy file (a file that lazy-loads your compo

nt out of the box and enables code splitting: <https://reactjs.org/docs/code-splitting.htm>

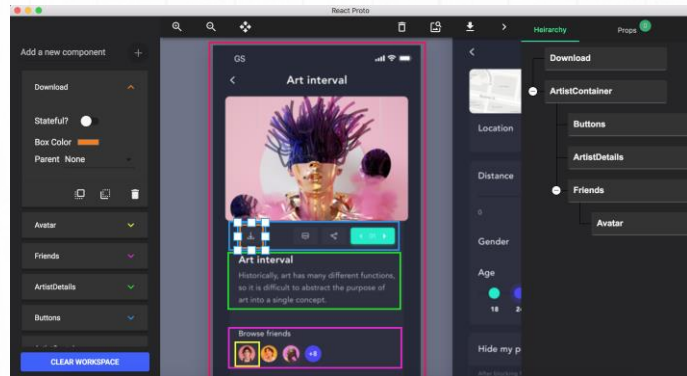
l#code-splitting) with each component you generate? Yes

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 a 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

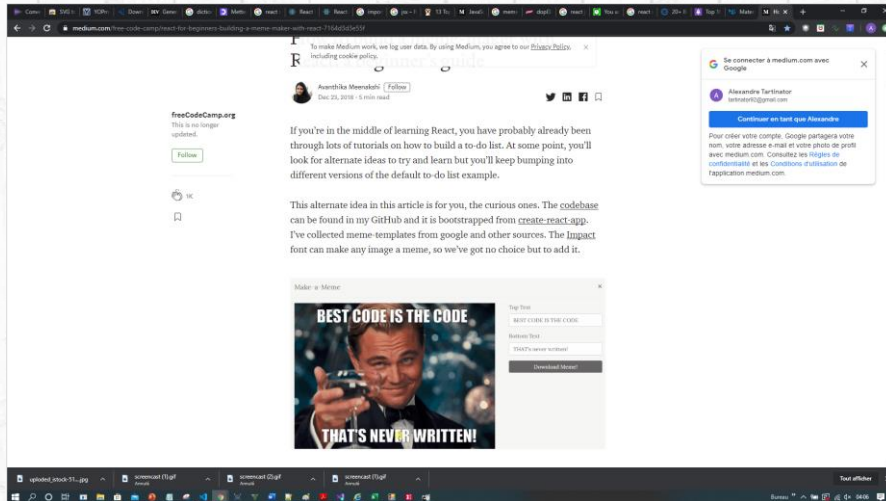
Commande :

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

• STRUCTURE GÉNÉRER :

```
~/dev/tmp $ cd my-dope-component/  
master* in ~/dev/tmp/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  
└── test/  
    ├── index.js  
    ├── styles.css  
    └── test.js  
  
4 directories, 14 files
```

TP1 : MEME generator



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

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

SVG



111



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 tramé
 - 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



SVG Définition

- 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 !)



SVG Définition

- 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/>

114

HOW TO : https://zlib.net/zlib_how.html

ZLIB : <https://zlib.net/>

Structure SVG

- 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...)

Exemple simple de code SVG :

```
<svg width="200" height="200" viewBox="0 0 500 500" version="1.1">
  <title> exemple SVG </title>
  <desc> Ceci est un exemple de SVG de base </desc>
  <defs></defs>
</svg>
```

svg & css

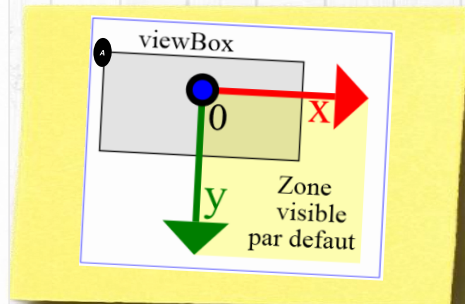
- La balise `<style type="text/css">`
- capacité `@media`
- sélection CSS classique $\frac{1}{2}$ & 3*si navigateur compatible
 - » `#premierGroup rect{ ... }`
- propriétés css pour déclarer un attribut svg
 - avec le même nom que l'attribut svg
 - exemple :
 - » en css : `fill : red;`
 - » en attribut svg : `fill="red"`

116



SVG & Système de coordonnées

- Point de référence et viewBox
- '0' références est le coins supérieur gauche
- Pas de 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 SVG de base :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <title> exemple SVG </title>
  <desc> Ceci est un exemple de SVG de base </desc>
  <circle r="100px" cx="350px" cy="350px" />
  <rect x="50" y="50" width="200" height="100" style="fill:red; stroke:blue; stroke-width:10" />
  <ellipse cx="100" cy="300" rx="50" ry="100" style="fill:#00ff00; stroke:black; stroke-width:1" />
  <line x1="400" y1="50" x2="350" y2="300" style="stroke:red; stroke-width:3" />
</svg>
```

Formes complexes SVG

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

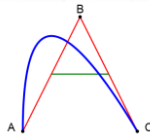
– **Le polygone / polylines** : forme pleine coordonnées infini

```
<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" />
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <title> exemple SVG </title>
  <desc> Ceci est un exemple de SVG de base </desc>
  <polygon points="170,170 220,170 220,220 170,220" style="fill:blue; stroke:black; stroke-width:1" />
  <polyline points="225,225 275,225 275,275 225,275" style="fill:none; stroke:black; stroke-width:1" />
  <path style="fill:red;opacity:0.5;stroke:black; stroke-width:1;" d="M100,100 H200
  C153 334 151 334 151 334
  C151 339 153 344 156 344
  C164 344 171 339 171 334
  C171 322 164 314 156 314
  C142 314 131 322 131 334
  C131 350 142 364 156 364
  C175 364 191 350 191 334
  C191 311 175 294 156 294
  C131 294 111 311 111 334
  C111 361 131 384 156 384
  C186 384 211 361 211 334
  C211 300 186 274 156 274" />
</svg>
```

Remarques :

Notez le code de l'expression d (data) de <path>, il s'agit de coordonnées de courbes de bézier (définition de points et tangentes) !

Remplissage et traits

- Remplissage **fill**

- Couleur **css**

fill="LIGHTBLUE"



- Gradient **svg**

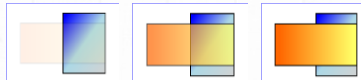
fill="url(#gradientId)"



- 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"



- Epaisseur

stroke-width="value"



120



Les Groupes SVG

- 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

👉 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>)
 - **<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 **<symbole 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" ... />**

122

Exemple de création et d'utilisation de symbole :

```
<defs>
<symbol id="croix" height="25" width="25">
  <g fill="red" stroke="none">
    <rect x="10" width="10" height="25"/>
    <rect y="10" width="25" height="10"/>
  </g>
</symbol>
</defs>
```

```
<use xlink:href="# croix " x="150" y="100"/>
<use xlink:href="# croix " x="250" y="0"/>
<use xlink:href="# croix " x="300" y="300"/>
```

Textures

- 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)

123

Exemples de gradient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <title> exemple SVG </title>
  <desc> Ceci est un exemple de SVG de base </desc>

  <defs>
    <linearGradient id="lingrad" x1="30%" y1="0%" x2="90%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0); stop-opacity:1"/>
      <stop offset="100%" style="stop-color:#FF0000; stop-opacity:1"/>
    </linearGradient>
  </defs>

  <circle r="100px" cx="350px" cy="350px" style="fill:url(#lingrad);"/>
  <rect x="50" y="50" width="200" height="100" style="fill:url(#lingrad); stroke:blue; stroke-width:10;"
/>
</svg>
```

Exemples de codes combinés :

http://www.w3schools.com/svg/svg_examples.asp

Pour connaître presque tous les éléments SVG :

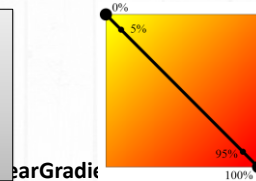
http://www.w3schools.com/svg/svg_reference.asp

Gradient SVG (dégradés)

- Dégradés

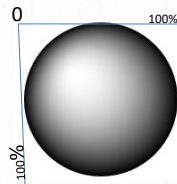
- Transition de couleur et/ou d'opacité est possible grâce aux gradient soit linéaire, soit circulaire (dit radiaux)

```
<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>
```



earGradient

- 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 SVG

- 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);"**

125

Exemples de filtres :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
```

```
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
```

```
xmlns="http://www.w3.org/2000/svg">
```

```
<title> Filtres SVG </title>
```

```
<desc> Ceci est un exemple de filtres SVG ! </desc>
```

```
<defs>
```

```
<filter id="blured">
```

```
<feGaussianBlur in="SourceGraphic" stdDeviation="5" />
```

```
<feOffset dx="5" dy="5"/>
```

```
</filter>
```

```
</defs>
```

```
<ellipse cx="100" cy="300" rx="50" ry="100" style="opacity:50%;fill:black; stroke:black; stroke-width:1;filter:url(#blured);"/>
```

```
<polyline points="225,225 275,225 275,275 225,275" style="fill:none; stroke:black; stroke-width:1;filter:url(#blured);"/>
```

```
</svg>
```

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..."**)

126

Exemple d'effet visuel de fondu :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <rect x="20" y="20" width="250" height="250" style="fill:blue">
    <animate attributeType="CSS" attributeName="opacity" from="1" to="0" dur="3s"
repeatCount="3" />
  </rect>
</svg>
```

Exemple de code d'animation :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <g transform="translate(80,150)">
    <text id="TextElement" x="0" y="0" style="font-family:Verdana;font-size:18"> C'est du SVG !
<animateMotion path="M 0 0 L 80 150" dur="5s" fill="freeze"/>
    </text>
  </g>
```

</svg>

Multiformats

- 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

127

Exemple de génération multi-formats :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" version="1.0" encoding="iso-8859-1" doctype-public="-//W3C//DTD XHTML 1.0
Transitional//EN" doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
  <xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:svg="http://www.w3.org/2000/svg" xml:lang="fr">
<head>
<title>GENERATION DE SVG ET XHTML EN XSLT</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<object id="AdobeSVG" classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"></object>
<xsl:processing-instruction name="import">
namespace="svg" implementation="#AdobeSVG"
</xsl:processing-instruction>
</head>
<body>
<h1>Chiffre d'affaires par livre</h1>
<svg:svg width="500" height="500" viewBox="0 0 500 500" version="1.1">
  <svg:title> SVG + XSL </svg:title>
  <svg:desc> Ceci est un exemple de SVG de base </svg:desc>
  <xsl:for-each select="bibliotheque/livre">
    <svg:rect x="100px" y="{20*position()}px" width="{@ca}" height="20" style="fill:red; stroke:black; stroke-
width:1">
      <svg:animate attributeName="width" attributeType="XML" begin="0s" dur="5s" fill="freeze" from="0"
to="{@ca}"/>
    </svg:rect>
    <svg:text x="10px" y="{20*position()+16}px" font-family="Arial" font-size="16px" font-
weight="bold"><xsl:value-of select="titre/text()"/></svg:text>
  </xsl:for-each>
</svg:svg> </body></html></xsl:template></xsl:stylesheet>
```