Module 307

Exercices

Version 2.1 du 25 mai 2021

Jean-Claude Stritt

Adaptations Pierre-Alain Mettraux



Table des matières

[1 Mise en place des infrastructures 1](#_Toc73440257)

[1.1 NetBeans 1](#_Toc73440258)

[1.2 Visual Code Studio 1](#_Toc73440259)

[1.3 FileZilla ou Total Commander 1](#_Toc73440260)

[1.4 Utilisation d'un modèle 1](#_Toc73440261)

[2 Rappel HTML5 (HTML+CSS3) + JavaScript et PHP 2](#_Toc73440262)

[2.1 Exercice 1 – base HTML5 (30’) 2](#_Toc73440263)

[2.2 Exercice 2 – login (90’) 2](#_Toc73440264)

[2.3 Exercice 3 – login avec PHP (60’) 6](#_Toc73440265)

[3 Apprentissage du JavaScript 9](#_Toc73440266)

[3.1 Exercice 4 – utilisation de jsfiddle (10’) FACULTATIF 9](#_Toc73440267)

[3.2 Exercice 5 – variables en JavaScript (60’) 10](#_Toc73440268)

[3.3 Exercice 6 – la structure de contrôle SWITCH (30’) 12](#_Toc73440269)

[3.4 Exercice 7 – les boucles en JavaScript (45’) 14](#_Toc73440270)

[3.5 Exercice 8 – tableau associatif ou tableau d’objets JSON (45’) 17](#_Toc73440271)

[3.6 Exercice 9 – démo sur les objets en JavaScript 20](#_Toc73440272)

[3.7 Exercice 10 – programmation orientée *« objets »* en JS (90’) 23](#_Toc73440273)

[3.8 Exercice 11 – programmation orientée *« classe »* en JS (90’) 27](#_Toc73440274)

[3.9 Exercice 12 – les closures – compteur (45’) 29](#_Toc73440275)

[3.10 Exercice 13 – les IIFE en JavaScript 31](#_Toc73440276)

[4 Apprentissage de jQuery 33](#_Toc73440277)

[4.1 Exercice 14 – bases de jQuery (90’) 33](#_Toc73440278)

[4.2 Exercice 15 – première utilisation de jQuery (90’) 34](#_Toc73440279)

[4.3 Exercice 16 – lecture jQuery & Bootstrap (facultatif) 37](#_Toc73440280)

[5 Web services 38](#_Toc73440281)

[5.1 Théorie (à lire, très important) 38](#_Toc73440282)

[5.2 Exercice 17 – test de web services (60’) 39](#_Toc73440283)

[5.3 Exercice 18 – application HTML5 d’appel à un web service (90’) 43](#_Toc73440284)

[5.4 Exercice 19 – application HTML5 avec double service web (45’) 46](#_Toc73440285)

[5.5 Exercice 20 – application *« login »* sur un serveur PHP (120’) 47](#_Toc73440286)

[5.6 Exercice 21 – application *« streetmap »* 48](#_Toc73440287)

# Mise en place des infrastructures

Les outils de développements sont essentiels pour être efficace. NotePad++ est utilisable pour dépanner, mais on va s'orienter sur des outils plus productifs. Je vous laisse choisir l'outil qui vous convient le mieux.

## NetBeans

L'IDE connu pas chacun de vous et qui permet aussi de développer du HTML. La version 12.x doit être utilisée (12.3 pour la dernière). Il a un plugin à rajouter c'est le Raimbow Braces pour les parenthèses en couleur.

## Visual Code Studio

C'est la Cadillac du développement web. Voici quelques plugins qu'il est bien d'avoir installés : Live Server, Bracket Pair Colorizer, JavaScript (ES6) code snippets, Prettier Code formatter, Material icon Theme, Jquery code snippets, CSS Peek et French Language. Si vous en avez d'autres de sympas, il n'est pas interdit de partager ☺.

## FileZilla ou Total Commander

Comme vous allez devoir héberger certains des exercices, un outil FTP est nécessaire. FileZilla n'est plus à présenter, mais j'ai un faible pour Total Commander, c'est le couteau suisse de l'informaticien développé par un ingénieur suisse.

Il serait bien de récupérer vos paramètres d'accès sur l'hébergement proposé par l'EMF. Si vous les avez paumés, vous pouvez vous mettre à 2 sur le même accès en ne mélangeant pas vos fichiers. Il est aussi possible d'installer un "WAMP server*»* pour faire du PHP. Avec ce logiciel, vous pouvez installer un *« virtual host »* (nommé *« 307 »* qui pointera sur votre dossier *« 307/exercices »*.

## Utilisation d'un modèle

J'ai préparé un dossier modèle (un template) qui sera utile pour les premiers exercices. Il faudra compléter les entêtes à chaque exercice. Vous pouvez adapter ce template comme bon vous semble.

# Rappel HTML5 (HTML+CSS3) + JavaScript et PHP

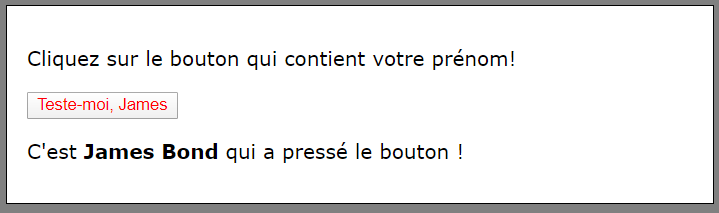
Les premiers exercices vont être utiles pour se remettre dans le bain

## Exercice 1 – base HTML5 (30’)

Pour ce premier exercice, veuillez créer ou copier, avec votre outil de développement, un projet *« ex01-html5-base »* dans le dossier principal *« 307/exercices » en partant du modèle* mis à disposition.

Adaptez le texte (votre nom et prénom) et la couleur en modifiant à bon escient les 3 fichiers à disposition. La ligne du bas s'affiche lors de l'appui du bouton.

### Maquette de la vue



### Code JS lancé à la fin du chargement de la page

**<body onload="initCtrl()">**

Si vous avez besoin d'initialiser certaines choses dans le JavaScript, mais que cela n'est possible que si la page est chargée (DOM créé entièrement), il est possible de lancer une fonction en fin de chargement. On peut aussi déplacer le lancement du script en fin de body au lieu du head. Observez la méthode **initCtrl()** qui ajoute un écouteur par programmation.

### HTML avec écouteur

Les lignes en commentaire sont identiques mais l’écouteur est en dur dans le code HTML, mais c’est à vous de l’ajouter ! Cette situation est très courante, mais il faut favoriser l’autre solution. Jouez avec les commentaires pour tester l’une ou l’autre solution.

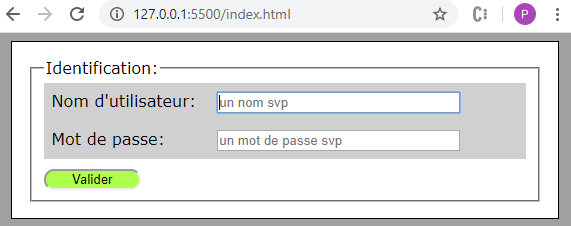
## Exercice 2 – login (90’)

Depuis votre outil de développement, veuillez copier l’exercice précédent *« ex01… »* vers *« ex02-html5-form-login »*.

On vous demande de compléter le conteneur pour afficher un composant formulaire (form) HTML5 pour simuler une opération de login. Pour le moment, le test du login se fera localement en JavaScript *(*valides : *« admin »* quelle que soit la casse pour l’utilisateur et *« emf123 »* pour le mot de passe).

Le but sous-jacent de cet exercice est de bien séparer les différents fichiers de la vue d’une application entre son **contenu** en HTML (des composants dans une page web), la **présentation** en CSS (le design) et la **logique** métier en JavaScript.

### Maquettes



|  |  |
| --- | --- |
| Si on introduit le nom d’utilisateur et le mot de passe correctement : | Si la validation n’est pas correcte (nom d’utilisateur ou mot de passe) : |

alert() ouvre un popup d'information, confirm() ouvre un popup de confirmation ou non et prompt() ouvre un popup d'input.

### Code HTML pour le formulaire

**<form class="user-form">**

**<fieldset>**

**<legend>Identification:</legend>**

**<div class="field">**

**<label for="username">Nom d'utilisateur:</label>**

**<input type="text" size="30" id="username"  
 placeholder="un nom svp" autofocus />**

**</div>**

**<div class="field">**

**<label for="username">Mot de passe:</label>**

**<input type="password" size="30" id="password"  
 placeholder="un mot de passe svp" />**

**</div>**

**<input type="button" value="Valider" id="valider" onclick="validerUtilisateur();">**

**</fieldset>**

**</form>**

Questions :

* À quoi sert l’attribut *« placeholder »* ?
* À quoi sert l’attribut *« autofocus »* ?
* Y a-t-il déjà du JavaScript dans ce code ?
* Quelle est la différence entre le bouton de l'ex01 et celui-ci ?

### Code CSS

Veuillez compléter *« main.css »* avec les classes ci-dessous pour obtenir un résultat égal à l’exemple de la vue de la page précédente:

* .user-form .field (padding et couleur de fond)
* .user-form .field label (inline-block et largeur)
* .user-form .field input (inline-block)
* #valider ou .user-form input[type=button] (largeur, rayon du bord, couleur de fond et marge)

### Code JS

Pour vous aider, les fonctions clés du JS sont ici :

* **document.getElementById("id")** 🡪 pour récupérer un élément du DOM si un **id** a été préalablement ajouté. Cette fonction avec la méthode **querySelector("…")** est la plus simple à utiliser car elles retournent un seul élément sur lequel il est ensuite facile de récupérer ou changer une propriété. L’accès aux propriétés se fait sans get et set mais en spécifiant « .value, .innertext, .innerhtml, … » par exemple.
* **console.log(…)** 🡪 pour afficher quelque chose dans la console du navigateur. Vous pouvez utiliser un littéral avec des « ` ` » (nouveauté ES6).
* **window.alert(…)** 🡪 pour afficher quelque chose dans un popup

À l’aide d’Internet, apprenez :

* Qu’est-ce que le DOM ?
* Comment définit-on une *« fonction »* (méthode) comme *« validerUtilisateur() »* ?
* Comment définit-on une *« variable »* (par exemple pour *« username »* et *« password »*) ?
* Les variables sont-elles *« typées »* (String par exemple) ?
* Où se trouve la console de débogage de Chrome ?

Vous complèterez les commentaires du fichier *indexCtrl.js* comme suit :

**/\***

**\* Contrôleur de la vue "index.html".**

**\***

**\* @author prénom + nom**

**\* @version 1.0 / date**

**\*/**

**​**

**function validerUtilisateur() {**

**// ...**

**}**

Puis *v*ous ferez la liaison avec ce dernier dans le code HTML. On peut faire cette liaison dans la partie *« head »* comme suit :

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<link rel="icon" href="#" />**

**<link rel="stylesheet" href="css/main.css">**

**<script src="js/indexCtrl.js"></script>**

**<title>Ex02 - login</title>**

**</head>**

Cette façon de charger le JS est ici appropriée, mais peut fortement ralentir la visualisation de la page si la ou les bibliothèques à charger sont conséquentes. Certains préconisent pour cette raison de plutôt charger les scripts après le chargement et visualisation de la page complète, c’est-à-dire juste avant le </body>. Mais HTML5 a aussi introduit l’attribut *«****async****»* à mettre dans la balise script. Cela indique au navigateur qu’aucune modification du DOM ne sera faite par ce chargement et que ce dernier se fera donc de manière asynchrone dans un thread séparé, donc sans soucis de lenteur au démarrage.

**Mais attention de charger synchrones et dans l’ordre les librairies nécessaires à d’autres.**

Pour terminer,

🡪 Prenez 5-10’ avec un camarade pour corriger le tout.

🡪 Résumez l’exercice avec les concepts clés dans votre RP.

## Exercice 3 – login avec PHP (60’)

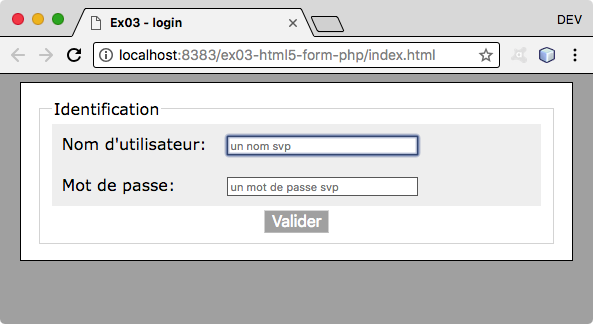
Depuis votre outil de développement, veuillez copier l’exercice précédent *« ex02… »* en *« ex03-html5-form-php »*. Le but est ici :

* D’améliorer un peu la vue en décolorant fortement la bordure du *« fieldset »* (lightgray) et en centrant et stylisant le bouton ;
* De faire la validation des informations de login par un petit script en PHP (au lieu du JavaScript) sur un serveur Apache à installer sur votre hébergement EMF. Il est aussi présent sur [**http://mettrauxpa.emf-informatique.ch**](http://mettrauxpa.emf-informatique.ch) avec les noms ci-joint.

Concrètement, vous allez répondre aux questions suivantes en testant chaque nouvelle partie :

1. Comment centrer le bouton de validation et le styliser ?
2. Comment envoyer le formulaire à un script PHP qui fera la validation ?

### Maquette de la vue



### Code HTML & CSS

Pour la 1re question, nous allons encapsuler la balise du bouton dans un <div>…</div> à qui nous ajoutons encore une classe *« button »*:

**<div class="button">**

**<input type="submit" value="Valider">**

**</div>**

Il suffit ensuite de styliser dans *« main.css »* la classe *« button »* (pour le centrage) et le bouton lui-même pour le type *« submit »* préféré à *« button »* (on vous explique de suite pourquoi) :

**.user-form .button {**

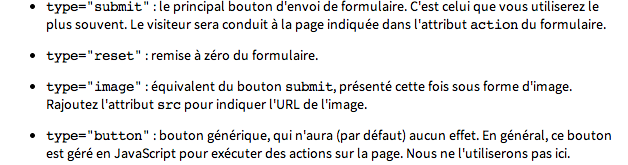
**}**

**.user-form input[type=submit] {**

**}**

Pour résoudre le 2e problème, vous aurez remarqué dans le code HTML ci-dessus que l’appel à la méthode de validation JavaScript *« validerUtilisateur() »*a été supprimé et que le type du bouton a aussi changé de *«****button****»* en *«****submit****»*. Cela indique au navigateur que le contenu du formulaire sera transféré via une méthode PHP indiquée dans la propriété *«****action****»* à ajouter à la balise *«****form****».*

Voici encore ce que nous résume *« OpenClassRooms »* pour les types de boutons :



C’est donc effectivement le type *«****submit****»* qu’il nous faut ici et le commentaire ci-dessus nous indique également le changement à faire dans la balise <form … >  en prenant soin d’indiquer l’URL complète pour l’action à exécuter :

**<form class="user-form" action="login.php" method="POST">**

**...**

**<input type="submit" value="Valider">**

**</form>**

Les propriétés HTML suivantes sont utilisées :

* **action** 🡪 permet d’envoyer le contenu du formulaire vers une page PHP (par exemple) avec le protocole HTTP(S) ;
* **method** 🡪 méthode HTTP(S) GET ou POST. Le GET est utilisé pour des demandes d’information et le POST pour des modifications/ajouts. Avec le GET, on est limité à 2048 caractères, URL incluse. Le défaut du GET est que l’on voit les informations directement dans l’URL.

### Script *« login.php »*

**<?PHP**

**// test si on a reçu une donnée de formulaire nommée "username"**

**if (isset($\_POST['username'])) {**

**// récupération des données transmises dans des variables locales**

**$username = strtolower($\_POST['username']);**

**$password = $\_POST['password'];**

**// affichage des infos reçues**

**echo "username: ".$username."</br>";**

**echo "password: ".$password."</br>";**

**​**

**// test username et mot de passe**

**if (($username == "admin") && ($password == "emf123")) {**

**echo "<script>alert('Validation OK');</script>";**

**} else {**

**echo "<script>alert('Utilisateur ou mot de passe incorrect !!!');</script>";**

**}**

**}**

**?>**

Pour que le code ci-dessus fonctionne, il faut rajouter des propriétés HTML « **name**="..." » aux champs correspondants (balises **INPUT** de type *«****text****»* ou *«****password****»*), car effectivement cela permet au PHP de retrouver les informations passées lors d’une requête HTTP(S) de type POST.

Questions:

* Par quoi commence et doit se terminer un code PHP ?
* Comment récupère-t-on une information dans le flux des informations GET ou POST ?
* Comment effectue-t-on une concaténation de chaines de caractères en PHP ?
* Avec quelle fonction intrinsèque (dans PHP) convertit-on une chaine en minuscules ?
* Quelle est la commande qui permet de renvoyer quelque chose vers le client ?
* Quel(s) autre(s) langage(s) trouve-t-on encore dans ce code PHP ?

Notes : Le code JavaScript de ce projet (indexCtrl.js) ne va plus servir à rien puisque la validation est faite maintenant dans le code PHP. Le concept de la *« session »* (sécurisation des futures transactions avec le serveur) sera abordé plus tard.

### Hébergement du code PHP et tests

Pour que *« login.php »* fonctionne, il vous faut :

* Un serveur Apache ;
* Le langage PHP installé sur ce serveur.

Vous avez alors deux possibilités assez simples :

1. Soit vous installez un produit gratuit comme WAMP, XAMPP ou MAMP (**option à privilégier**). Ces produits installent Apache, PHP et MySQL sur votre propre machine et permettent de s’en servir comme un serveur standard tout à fait valide.
2. Soit vous travaillez avec votre site *«  monnom.emf-informatique.ch »* qui est justement hébergé sur un serveur Apache avec le langage PHP installé ;

L’URL dans la propriété *« action »* de la balise <form> pourrait alors être (par exemple) :

http://locahost/307/exercices/login.php (WAMP ou XAMPP)

http://localhost:8888/307/exercices/login.php (MAMP)

http://monnom.emf-informatique.ch/307/exercices/login.php (EMF-INFORMATIQUE)

Pour terminer

🡪 Prenez 5-10’ avec un camarade pour corriger le tout.

🡪 Résumez l’exercice avec les concepts clés dans votre RP.

Pour les rapides:

L'exemple ci-dessus utilise un POST que l'on utilise pour les ajouts. On doit dans ce cas utiliser un GET, car c'est de la lecture pure. Transformez votre index.html et le PHP pour faire fonctionner votre page et votre site avec un GET.

# Apprentissage du JavaScript

Dans les exercices précédents, vous avez appris à utiliser quelque peu JavaScript comme suit :

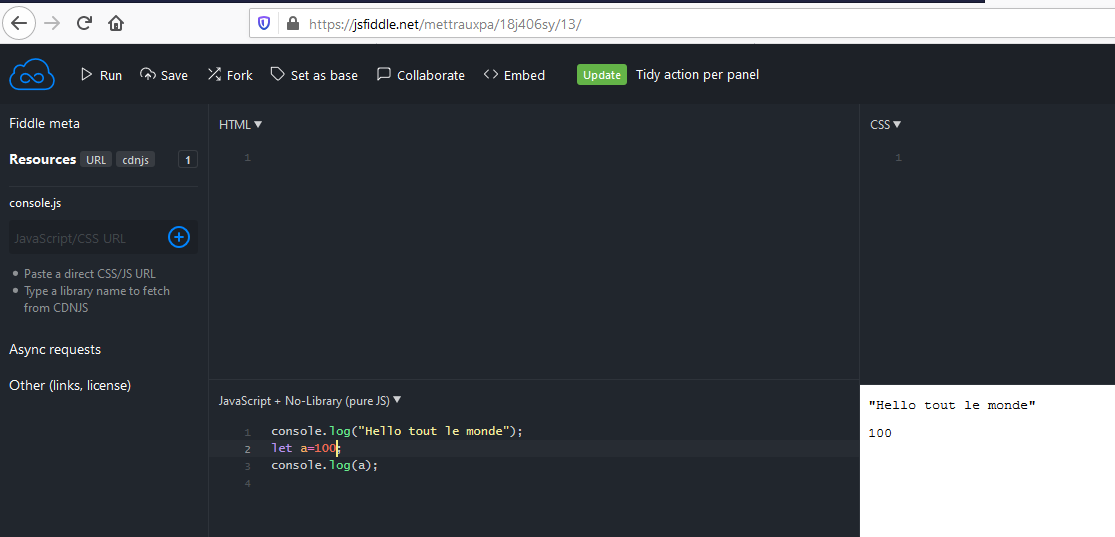
* Créer une fonction qui ne retournait rien ;
* Récupérer des informations dans le DOM (l’arbre des nœuds d’une page web) ;
* Créer quelques variables et stocker de l’information dedans ;
* Utiliser une *« structure de contrôle »* comme un IF ;
* Afficher de l’information dans la console ou dans un popup.

Il reste beaucoup de travail d’apprentissage, car même si les *« structures de contrôle »* (if, for) de JS ressemblent à celles de Java, on y trouve même les classes depuis ES6. Il y a encore du travail, mais nous verrons que nous pourrons passer par-dessus chaque difficulté (je l’espère ☺) avec quelques astuces et du travail.

## Exercice 4 – utilisation de jsfiddle (10’) FACULTATIF

Le site <http://jsfiddle.net/> propose un outil très simple pour tester du JavaScript à partir de code HTML + CSS. Le tout est appelé un *« fiddle »* que l’on peut traduire en français par une *« combine »* (ou une astuce). Je vous conseille donc :

* De créer un compte gratuit chez eux (pour pouvoir sauver ou partager des fiddle)
* De tester un premier fiddle avec un affichage du fameux *« hello world »* dans un popup JavaScript (ce que vous savez déjà faire) :



Vous pouvez noter le codage très souple, puisqu’aucune balise HTML n’a précisé que nous utilisions du JavaScript. Vous auriez aussi pu utiliser *« console.log("Hello world !") »* pour afficher le même message dans la console de *« jsfiddle »*, mais avant cela, il faut ajouter une petite dépendance (sous *« Resources »*) sur un code qui ajoute la console à *« jsfiddle »* :

|  |  |
| --- | --- |
|  | <https://rawgit.com/eu81273/jsfiddle-console/master/console.js> |

## Exercice 5 – variables en JavaScript (60’)

Vous allez taper des commandes en JavaScript. Vous allez utiliser la console de votre navigateur ou bien le *« jsfiddle »* de l'exercice précédent.

Pour déclarer une variable, vous avez 4 possibilités :

1. myVar;

2. **var** myVar;

3. **let** myVar;

4. **const** myVar=1;

Les points 1 et 2 sont plus ou moins équivalents, mais il ne faudrait plus les utiliser, car cela laisse trop de liberté (possibilité de déclarer 2x la même variable). Le var est utilisé depuis la nuit des temps et vous trouvez beaucoup d'exemples dans la documentation. Beaucoup de programmeurs sont toujours bloqués dessus.

Le point 1 est utilisé pour des variables globales, on le verra dans les derniers exercices.

**Le let est le mot-clé qu'il faut utiliser.**

Le **const** est aussi utile pour les constantes et les objets, car on peut toujours changer les propriétés.

1. Effacer le contenu de la console avec *« console.clear() »* ;
2. Créer une variable nommée *« a »*;
3. Afficher le contenu de *« a »*avec console.log;
4. Stocker la valeur 15 dans cette variable ;
5. Afficher le contenu de cette variable dans la console sous la forme *« Ma variable a = ? »*
6. Créer une variable nommée *« b »* et lui assigner directement la valeur 9 ;
7. Afficher le contenu de cette variable dans la console sous la forme *« Ma variable b = ? »*
8. Faire l’addition de ces 2 variables en affichant directement le résultat dans la console sous cette forme : « 15 + 9 = ? » ; (essayer d’utiliser un littéral avec `…${…}…`)
9. Compléter en faisant de même pour une soustraction, une multiplication et une division des deux variables ;
10. Stocker *« Bonjour »* dans la variable a ;
11. Stocker *« les amis »* dans la variable b ;
12. Afficher *« bonjour les amis »* dans la console en concaténant les variables ;
13. Stocker *« true »* dans la variable a ;
14. Stocker *« false »* dans la variable b ;
15. Effectuer une opération AND entre les 2 variables et afficher le résultat sous cette forme *« true AND false = ? » ;*
16. Effectuer une opération OR entre les 2 variables et afficher le résultat sous cette forme *« true OR false = ? » ;*
17. Stocker la date du jour dans la variable a avec new Date();
18. Calculer une nouvelle date dans la variable b qui est 61 jours avant la date courante (utilisation getDate, setDate)
19. Afficher les dates suivante en vous aidant de *« toLocaleString() et toLocaleDateString() »* :

**12/05/2020 à 17:05:57**

**12/05/2020**

**29/02/2020 à 16:56:06 // il y a 61 jours**

**12.05.2020 // format Suisse romand**

À ce stade, nous pouvons dire que les variables JavaScript ne sont pas typées et peuvent contenir :

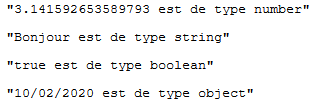
- des nombres (**number**);

- des chaines de caractères (**string**) ;

- des booléens (**boolean**) ;

- des objets (**object**)

Le mot réservé *«****typeof****»* permet de connaitre le type utilisé momentanément pour une variable. Exemple : console.log(typeof a);

1. Stocker la valeur de Math.PI dans a, *« bonjour »* dans b, créer et assigner true dans c, créer et assigner la date courante dans d, puis afficher le type pour les 4 variables :  
   
2. Pour terminer : mettre le code et tous les résultats dans le RP.

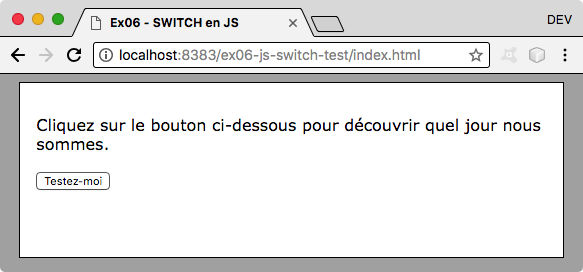
## Exercice 6 – la structure de contrôle SWITCH (30’)

Depuis votre outil de développement, copier l’exercice *« ex01… »* vers *« ex06-js-switch-test »*.

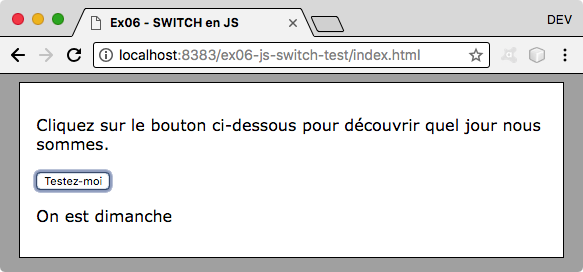
Nous avions déjà testé le *« IF »*, testons maintenant le *« SWITCH »*, avec une petite application qui doit afficher le jour de la semaine directement en remplacement du contenu d’une balise HTML existante.

### Maquettes de la vue

Au démarrage de l’application :



Après le clic, affichage du jour de la semaine :



### Code HTML

**<body >**

**<div id="container">**

**<p>  
 Cliquez sur le bouton ci-dessous pour découvrir quel jour nous sommes.  
 </p>**

**<button onclick="afficherJourSemaine();">Testez-moi</button>**

**<p id="info">&nbsp;</p>**

**</div>**

**</body>**

### Code JavaScript

Il s’agit de créer une fonction *« afficherJourSemaine() »* dans le fichier annexé qui va indiquer à l'opérateur le jour de la semaine du jour courant. Une méthode de la classe Date retourne le jour de la semaine.

Pour terminer : 🡪 mettre dans le RP, le code JS et une explication sur le concept sous-jacent de remplacement du contenu d’une balise HTML par de l’information gérée en JavaScript

### Tableau

Au lieu de mettre toute une série de if ou un switch, il est possible d'utiliser un tableau. Un tableau fonctionne sur le même principe qu'un tableau Java. La grande différence est que l'on peut faire grandir ou rétrécir un tableau JS.

**const tab = [12, 45, 56];**

**const noms = ["Alice", "Bob", "Clara", "Dave"];**

**console.log(noms[0]); // affichage d'Alice**

**console.log(noms.length); // affichage de 4**

**noms.push('Eric'); // ajoute Eric à la fin du tableau**

**noms.unshift('AA'); // ajoute AA au début du tableau**

**noms.pop(); // efface le dernier élément du tableau**

**noms.shift(); // efface le premier élément du tableau**

**for (let i=0; i<noms.length; i++) {**

**console.log(i + " : " + noms[i]); // 0 : Alice …**

**}**

**for (let i in noms) {**

**console.log(i + " : " + noms[i]); // 0 : Alice …**

**}**

**noms.forEach(function(nom) { // Alice …**

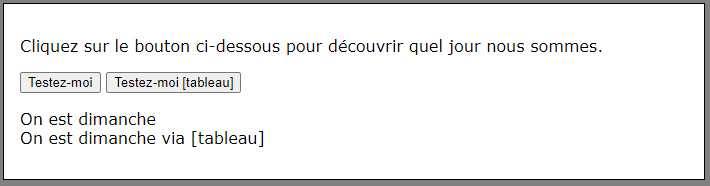
**console.log(nom);**

**});**

**noms.forEach(nom => console.log (nom)); // Alice …**

**noms.forEach((nom,i) => console.log (i + " : " + nom)); // 0 : Alice …**

Pour les rapides : faites la même chose avec un tableau contenant les 7 jours de la semaine (ajoutez un bouton et créez un "div" dans le paragraphe « info » dans le JS ou utiliser le même paragraphe)

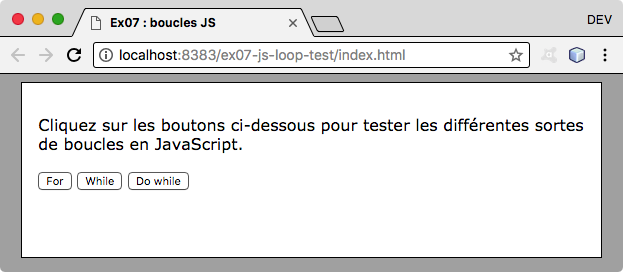


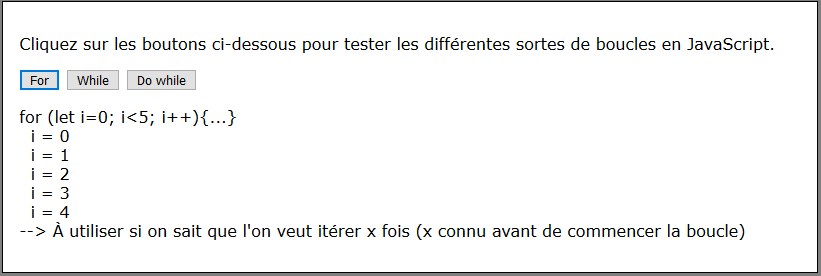
## Exercice 7 – les boucles en JavaScript (45’)

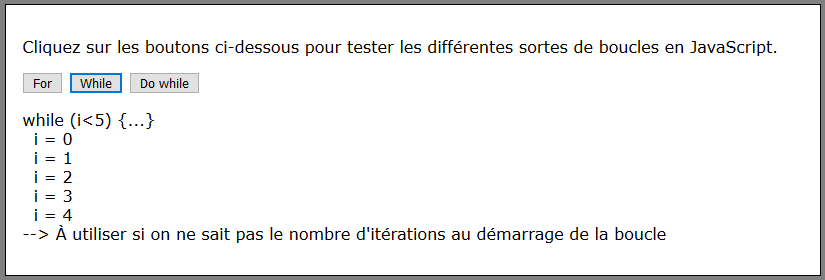
Depuis votre outil de développement, veuillez copier *« ex06… »* vers *« ex07-js-loops-test »*.

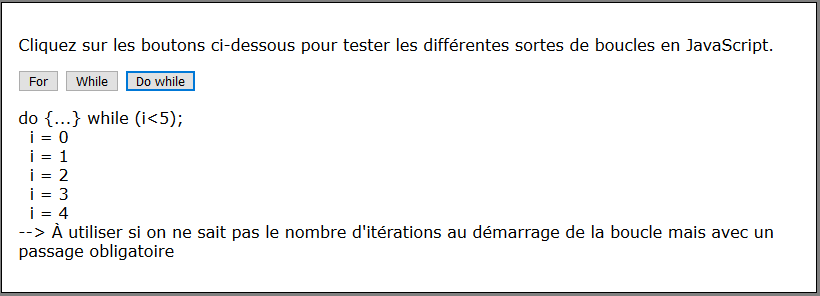
Nous allons tester toutes les sortes de boucles possibles en JavaScript avec une application qui ressemble à la précédente du point de vue de l’insertion d’informations dans du HTML par l’entremise de JavaScript.

### Maquettes de la vue









### Code HTML

**<body onload="initCtrl()">**

**<div id="container">**

**<p>Cliquez sur les boutons ci-dessous pour tester**

**les différentes sortes de boucles en JavaScript.</p>**

**<button id="for">For</button>**

**<button id="while">While</button>**

**<button id="do">Do while</button>**

**<p id="info">&nbsp;</p>**

**</div>**

**</body>**

### Code JavaScript

Il s’agit ici de créer trois fonctions différentes :

* testerFor() ;
* testerWhile() ;
* testerDoWhile().

… à mettre soit dans une balise <script> … </script>, soit dans un fichier annexé *« IndexCtrl.js »* que vous stockez dans le dossier *« js »* (il faut alors faire la liaison dans le HTML). Ces méthodes font de la concaténation de chaines de caractères (fonctionnement identique à Java).

### Évènements

On peut gérer les évènements JavaScript de 3 manières:

1. Via des attributs (plus recommandé), mais encore bien présent dans ce document)

**<button onclick="testerFor();">For</button>**

1. Via des propriétés du JavaScript assez semblable au pt 1. Cela n'autorise qu'un appel de fonction par type d'événement

**<button id="for">For</button>**

**let but = document.getElementById("for");**

**but.onclick = testerFor; // appel de la function testerFor**

1. Via la méthode **addEventListerner** (recommandé) . Il est possible de d'ajouter plusieurs fonctions et il est possible d'enlever un écouteur via la méthode removeEventListener

**<button id="for">For</button>**

**let but = document.getElementById("for");**

**but.addEventListener("click", testerFor); // appel de la function testerFor**

**// ou sur une ligne**

**document.getElementById("for").addEventListener("click", testerFor);**

Pour cet exercice, utilisez la manière no 3, mais comme le script est chargé avant la création du DOM, il peut générer des erreurs avec des éléments pas encore créés. Une possibilité est de mettre un écouteur sur le <body onload="…"> avec une fonction d'initialisation du JS.

Pour terminer : 🡪 Mettre dans le RP, le code HTML + JS en commentant encore le concept des boucles (comparaison avec Java), si les commentaires du code ne sont pas encore assez explicites.

## Exercice 8 – tableau associatif ou tableau d’objets JSON (45’)

Depuis votre outil de développement, veuillez copier *« ex07… »* vers *« ex08-js-json-array-loop »*. Supprimez tout le code JavaScript, pour le reconstituer avec les informations suivantes :

Nous allons tester une **imbrication de 2 boucles** sur les données d’un tableau de personnes défini par un objet de type JSON (JavaScript Object Notation).

JSON est la structure de données par excellence traitée de manière prioritaire en JS, car …

* Elle est compréhensible immédiatement aussi par un humain.
* Elle est indépendante des langages informatiques (même si elle est tirée de la notation de création des objets en JavaScript).
* Elle permet de stocker plusieurs types de données
* Elle propose une structure en arborescence
* Il y a une très grande affinité entre un objet JavaScript et celui transite par le réseau.

Nous y reviendrons fréquemment dans plusieurs exercices. Voici un exemple :

**const personne = {**

**prenom: "Jules",**

**nom: "Tartampion",**

**age: 25**

**};**

**// ou bien**

**const personne = {**

**"prenom": "Jules",**

**"nom": "Tartampion",**

**"age": 25**

**};**

**​**On utilise l’opérateur *«****in****»* dans un *«****for****»* pour balayer tous les champs *«****f****»* (field) de l’objet. Exemple :

**for (let f in personne) {**

**console.log("field: " + f + ", valeur: " + personne[f]);**

**}**

**Résultat :**



Le JSON peut aussi contenir un tableau d’objets (ici des personnes) :

**// définition d'un objet avec un tableau de personnes (A) A utiliser dans l’exercice**

**const json = {**

**personnes: [**

**{prenom: "John", nom: "Doe", age: 44},**

**{prenom: "Anna", nom: "Smith", age: 32},**

**{prenom: "Peter", nom: "Jones", age: 29}**

**]**

**};**

**// définition d'un tableau de personnes (B) (autre façon de faire)**

**const personnes = [**

**{prenom: "John", nom: "Doe", age: 44},**

**{prenom: "Anna", nom: "Smith", age: 32},**

**{prenom: "Peter", nom: "Jones", age: 29}**

**] ;**

… que l’on parcourt (A) alors avec une boucle FOR standard :

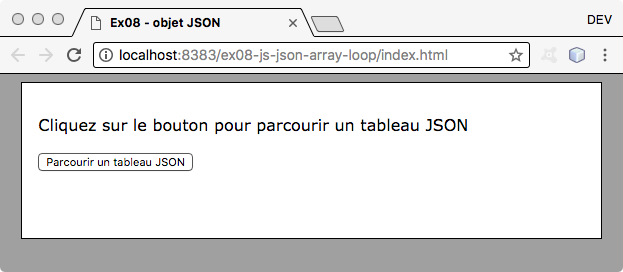
**for (let i = 0; i < json.personnes.length; i++) {**

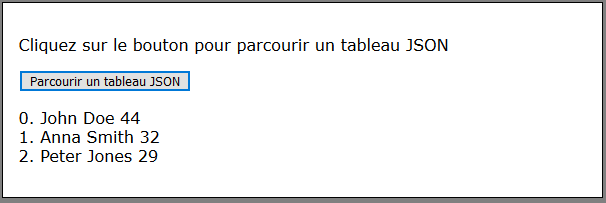
**let personne = json.personnes[i];**

**// ...**

**}**

### Maquettes de la vue





### Code HTML

**<body>**

**<div id="container">**

**<p>Cliquez sur le bouton pour parcourir un tableau JSON</p>**

**<button onclick="parcourirUnTableauJSON();">  
 Parcourir un tableau JSON  
 </button>**

**<p id="info">&nbsp;</p>**

**</div>**

**</body>**

### Code JavaScript

Il s’agit ici de créer la fonction *« parcourirUnTableauJSON() »* dont le but est de concaténer dans une variable *« txt »* tout le contenu d’un tableau JSON.

Notez que les données et les éventuelles méthodes *« métier »* pourraient être déplacées dans un fichier *« worker.js »* pour faire du MVC côté *« vue »*. Nous ne le ferons pas encore ici, mais dans l’exercice suivant.

Pour terminer : 🡪 mettre dans le RP, le code HTML + JS en commentant encore le concept du JSON, grâce aux explications de ce site :

<http://blog.idleman.fr/le-json-un-pont-entre-tous-les-langages/>

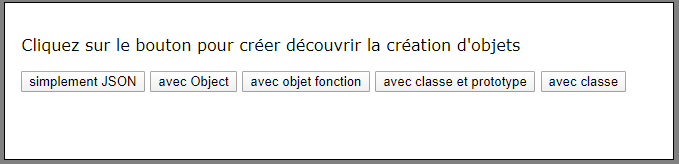
Pour les rapides : créez un deuxième paragraphe « infoBis ». Il est possible d’utiliser une arrow function (fonction flèche) en utilisant l’instruction « forEach » et faite en sorte que l’affichage soit identique.

## Exercice 9 – démo sur les objets en JavaScript

Le maitre mot de cet exercice est l'observation. Il suffit de récupérer l'exercice et de tester les différentes possibilités. Dans le gros exercice 10 qui nous attend, on va utiliser un bean Personne. Il y a plusieurs manières de créer un bean, cela va de l'objet JSON à la création d'une classe pour ensuite instancier un ou plusieurs beans.

### Bean *« Personne »*

Dans l’exercice précédent, nous avons vu comment créer un objet avec la notation JSON. Il y a une 2e façon de faire en créant une instance à partir de la fonction *« Object »*. Ensuite, on peut définir des classes pour créer des objets, on a même 3 manières de faire. Le dossier *« ex09-js-poo-demo-objet » présente les différentes manières de faire avec affichage dans la console.*



Résumons la création des objets avec les 2 méthodes sans utiliser des classes:

**Créer un objet en JSON (JavaScript Object** N**otation)**

**const p1 = {**

**prenom: "Jules",**

**nom: "Tartampion",**

**age: 25**

**};**

**p1.ville = "Fribourg" ; // ajoute un attribut à p1**

**Créer un objet avec la fonction Object (très rare)**

|  |  |  |
| --- | --- | --- |
| **const p2 = new Object(); p2["prenom"] = "Juliette"; p2["nom"] = "Tartampion"; p2["age"] = 23;** | **ou** | **const p2 = new Object(); p2.prenom = "Juliette"; p2.nom = "Tartampion"; p2.age = 23;** |

Comment cela est-il possible avec cette fonction *« Object »* ?

* JS gère les attributs d’un objet comme des éléments d’une *« hash map »* (liste de données de type clé-valeur). On peut ainsi ajouter, modifier ou supprimer un élément à tout instant.
* Autre particularité que nous avons déjà vue : puisque les variables ou propriétés sont non typées au départ, on peut y stocker n’importe quoi et même une **fonction stockée alors comme un objet**. Cette particularité nous permettra de *« surcharger »* des méthodes existantes dans un objet ou d’en rajouter.

Voyons justement l’affichage de ces deux objets p1 et p2 par *« console.log »*, par *« alert() »* ou en remplissant du HTML : cela n’est guère convaincant puisque le résultat donne ceci :



Comment peut-on donc surcharger les 2 méthodes « **toString**» qui nous affichent ce résultat ? Très simplement en remplaçant l’attribut nommé *« toString »* (qui contient une fonction) comme suit (notez les 2 notations possibles) :

**p1.toString = function() {  
 return this.prenom + " " + this.nom;  
};**

**p2["toString"] = function() { // très rare  
 return this.prenom + " " + this.nom;  
};**



Cette manière de faire implique qu’il faut ajouter à chaque objet la fonction (méthode) toString ainsi que d’autres fonctions si l’on désire créer un objet complexe (on est encore loin de la classe Java !)

**Créer un objet avec la fonction objet**

Il est possible de créer une « Function Objects » fonction objet. On s’approche gentiment de la classe style Java. Le fonctionnement est identique au « new Object() » vue à la page précédente.

**function Personne(prenom, nom, age) {  
 this.prenom = prenom;  
 this.nom = nom;  
 this.age = age;**

**this.toString = function(){**

**return this.nom + " " + this.prenom + " (" + this.age + ")";**

**}  
}**

On peut maintenant instancier un objet avec le **new,** mais on a la même problématique : la fonction toString fait partie de l’objet et est donc dupliquée pour chaque objet. Il est possible d'ajouter des fonctions par après.

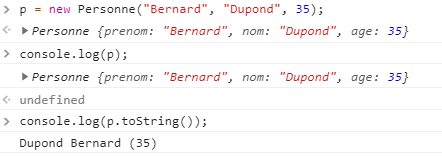


**Créer un objet avec la classe et le prototype**

Pour éviter la duplication des fonctions, if faut utiliser la programmation par **prototypage** grâce à l’attribut *« prototype »* qui se trouve dans tout objet JS. Voici un exemple et cette fois la fonction a disparu de l’objet, on s’approche de ce qui est fait en Java :

**function Personne(prenom, nom, age) {  
 this.prenom = prenom;  
 this.nom = nom;  
 this.age = age;**

**}  
  
Personne.prototype.toString = function() {  
 return this.nom + " " + this.prenom + " (" + this.age + ")";  
};**



Nous venons de définir une pseudo-classe (à la Java) nommée *« Personne »*, dans laquelle nous avons surchargé la méthode par défaut *« toString »*. **Vous pouvez dès lors copier ce code dans le fichier *« personne.js »***.

**Créer un objet avec la classe**

Introduit avec ES6 en 2015, la voici la dernière manière pour créer une classe à la mode Java. On utilisera cette façon de faire dans la suite de l'exercice.

**class Personne {**

**constructor(prenom, nom, age) {  
 this.prenom = prenom;  
 this.nom = nom;  
 this.age = age;**

**}**

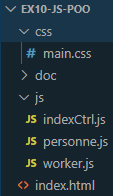
**toString() {**

**return this.nom + " " + this.prenom + " (" + this.age + ")";**

**}  
}**



## Exercice 10 – programmation orientée *« objets »* en JS (90’)

Depuis votre outil de développement, veuillez copier l’exercice *« ex03… »* vers *« ex10-js-poo »*. Supprimez tout le code JavaScript, pour le reconstituer plus tard avec les informations qui vont suivre. Cet exercice est l’un des plus importants de l’apprentissage de JavaScript, car il concerne la *« programmation orientée objets »* (POO) et la structuration de l’application en *« modules-fichiers »* de type MVC.

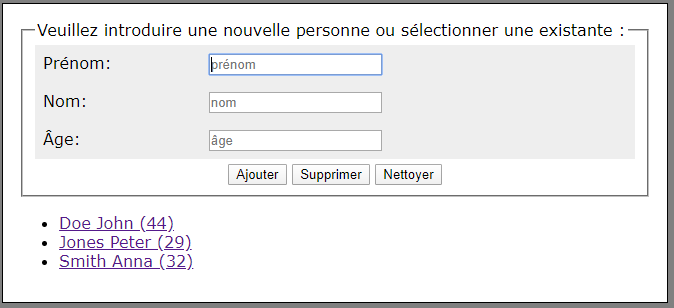
Le but final est ici de construire une application qui permette de gérer simplement les informations d’une liste de personnes (prénom, nom, âge) avec des opérations *« métier »* comme *« ajouter une personne »* ou *« supprimer une personne »* que nous mettrons dans un fichier *«****worker.js****»*. Pour gérer des personnes, il faut également disposer d’un bean *« Personne »* (à la Java) que nous stockerons dans un fichier *«****personne.js****»*.

**Vous pouvez dès à présent créer ces 2 fichiers JavaScript, mais vides pour le moment.**

Nous ferons toute la partie *« contrôle »* de l’affichage et les interactions utilisateur dans le contrôleur nommé *«****indexCtrl.js****».*

### Maquette principale de la vue

Au démarrage, la vue affichera une liste initiale triée de 3 personnes :



Si des noms sont ajoutés, la liste sera réaffichée avec les noms triés (fonction *« .sort »*) :



La liste affichée utilise un lien pour chaque élément avec href= "#" et un onclick. Il est aussi possible de laisser les éléments de liste normaux avec aussi un écouteur, par contre, il est visuellement important de changer le curseur lorsqu’on passe au-dessus des éléments de la liste.

### Code HTML

Le code HTML doit charger les différents *« modules-fichiers »* de notre application. Vous devrez les intégrer dans la partie *« <head> … </head> »*. Les trois scripts sont ainsi exécutés et les variables et les fonctions sont accessibles au chargement de la page. Comme le worker initialise des personnes, le script personne doit être placé avant, dans l'ordre d'exécution.

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="icon" href="#">

<link rel="stylesheet" href="css/main.css" >

<title>Ex10 - POO - prototype</title>

<script src="js/personne.js"></script>

<script src="js/worker.js" ></script>

<script src="js/indexCtrl.js" ></script>

</head>

Nous devons transformer le formulaire comme suit, mais sachez d’ores et déjà que :

* La balise *« form »* n’a plus besoin de la propriété *« action »* vers une page PHP, car nous n’avons plus de PHP à gérer.
* Les propriétés *« name »* des balises *« input »* peuvent être modifiées en *« id »,* car nous n’avons plus à envoyer une requête HTTP de type GET ou POST, mais nous devrons récupérer ces informations dans le contrôleur avec des *« document.getElementById(id).value ».*
* Il n’y aura plus de bouton de type *« submit »* non plus*,* car encore une fois, nous n’avons plus de formulaire à envoyer par HTTP(S).

<body>

<div id="container">

<form class="user-form">

<fieldset>

<legend>Veuillez introduire une nouvelle personne   
 ou sélectionner une existante&nbsp;:</legend>

<div class="field">

<label for="prenom">Prénom:</label>

<input type="text" size="20" id="prenom"

placeholder="prénom" required="required" autofocus>

</div>

<div class="field">

<label for="nom">Nom:</label>

<input type="text" size="20" id="nom"

placeholder="nom" required="required">

</div>

<div class="field">

<label for="nom">Âge:</label>

<input type="text" size="20" id="age"

placeholder="âge" required="required">

</div>

<div class="button">

<input type="button" value="Ajouter" onclick="ajouter();">

<input type="button" value="Supprimer" onclick="supprimer();">

<input type="reset" value="Nettoyer">

</div>

</fieldset>

</form>

<div id="info"></div>

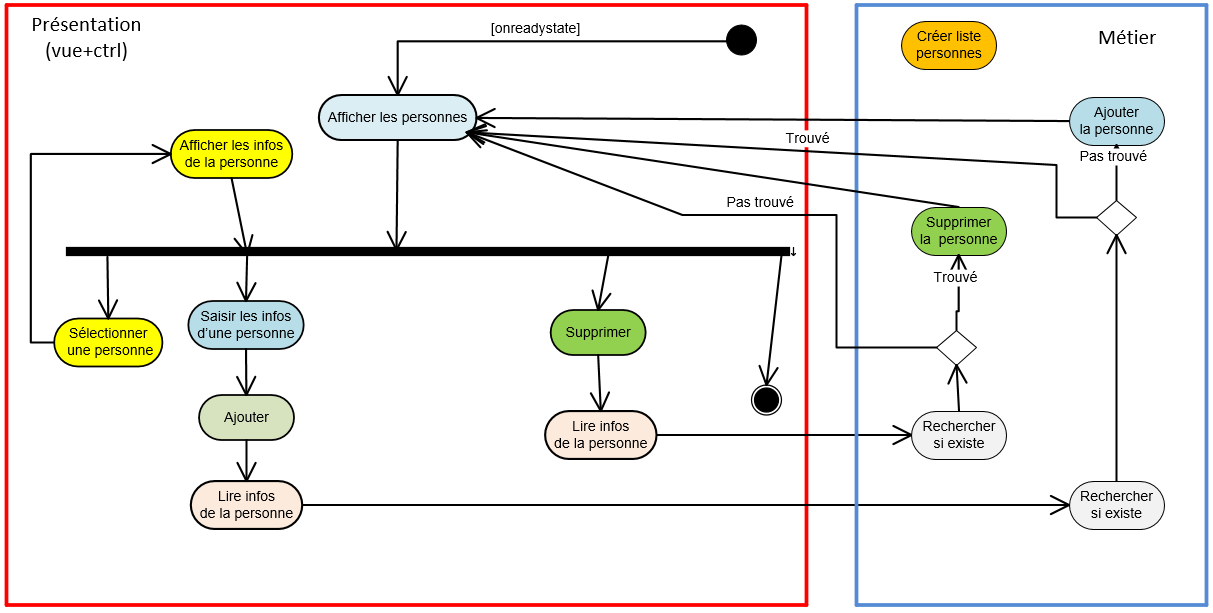
</div>

</body>

Notez les 2 gestions d’évènements *«****onclick****»* qui appelleront des fonctions (méthodes) du contrôleur. Nous programmerons ces fonctions plus tard. Notez également le calque *«****info****»* (div) qui contient au départ un *« blanc insécable »* (**&nbsp;**) qui sera remplacé au final par la liste des personnes dans une structure **<ul> … </ul>**.

### Diagramme d’activité

Il est maintenant temps de programmer la partie *« métier »* (actions à droite du schéma), puis la partie *« présentation »* avec les actions dans le contrôleur (à gauche) :



### Bean dans *« personne.js »*

Comme en Java, on fait des beans avec des propriétés et des méthodes. Ici pas de get/set car il n'existe pas encore de propriété privée (mais cela arrive !). C'est une fonction prototype avec 3 attributs (nom, prenom, age) qu'il faut écrire avec, en plus, une fonction (prototype) toString(). Chaque fois que vous utilisez un attribut dans le bean, vous devez le précéder par le mot-clé **this**.

function Personne(prenom, nom, age) {

this.prenom = prenom ;

...

}

Personne.prototype…

### Couche *« métier »* dans *« worker.js »*

Pour ajouter ou supprimer une personne de la liste, on aura besoin d’une fonction private ***« \_trouverPersonne(p)****»*. En effet, si une personne doit être ajoutée à la liste, on contrôlera qu’elle n’existe pas encore. Si une personne doit être supprimée de la liste, on contrôlera qu’elle existe.

Cette fonction *« \_trouverPersonne »* peut retourner l’index (idx=0, 1, 2 ...) dans la liste de la personne recherchée ou éventuellement -1 si elle

n’a pas été retrouvée. Cela permettra de la supprimer facilement avec la fonction *« .****splice(idx, 1)****»* disponible sur les tableaux. La fonction *« .****push(p)****»* permettra d’ajouter une personne à la liste.

// définition du modèle de données, créé au chargement du js dans le index.html

const personnes = [

new Personne("John", "Doe", 44),

new Personne("Anna", "Smith", 32),

new Personne("Peter", "Jones", 29)

];

// premier tri de la liste de personnes

personnes.sort();

// fonction privée pour retrouver l'index d'une personne dans le tableau, -1 autrement

// il faut comparer avec **toString()**

function \_trouverPersonne(p) {

let idx = -1;

for (let i=0… {

...

}

return idx;

}

// ajouter une personne dans la liste des personnes si pas trouvée

function ajouterPersonne(p) {

let idx = \_trouverPersonne(p);

...

}

// supprimer une personne dans la liste des personnes si trouvée

function supprimerPersonne(p) {

let idx = \_trouverPersonne(p);

...

}

### Contrôleur dans *« indexCtrl.js »*

Un peu comme en Java FX (module 120), ce contrôleur peut être divisé en 3 parties :**​**

/\*

\* 1. DOM PRET : DEMARRAGE DE L'APPLICATION

\*/

document.**onreadystatechange** = function () {

if (document.readyState === "complete") {

**\_afficherPersonnes**();

}

};

/\*

\* 2. METHODES PRIVEES DE LECTURE/ECRITURE DANS LA VUE

\*/

// affiche la liste des données au bas de la vue (avec du HTML généré)

function **\_afficherPersonnes**() {

let txt = "<ul>";

...

txt += "</ul>";

document.getElementById("info").innerHTML = txt;

}

// affiche les infos d'une personne dans le formulaire

function **\_afficherInfosPersonne**(p) {

...

}

// lit le contenu des masques de saisie pour en faire une personne

function **\_lireInfosPersonne**() {

let p = null;

...

return p;

}

/\*

\* 3. METHODES PUBLIQUES NECESSAIRES A LA VUE

\*/

// appelée depuis la vue pour afficher les données de la personne sélectionnné

function **selectionnerPersonne**(i) {

\_afficherInfosPersonne(personnes[i]);

}

// appelée depuis la vue pour ajouter une personne

function **ajouter**() {

let p = **\_lireInfosPersonne**();

...

}

// appelée depuis la vue pour supprimer une personne

function **supprimer**() {

let p = **\_lireInfosPersonne**();

...

}

## Exercice 11 – programmation orientée *« classe »* en JS (90’)

### Une classe en JS

Depuis la sortie de ES6 en 2015, on peut aussi utiliser des classes style Java avec un constructeur. Il même possible de faire de l’héritage avec le mot extends. Les aficionados de Java ne seront pas totalement perdus tout en sachant que les classes JS n’offrent pas les mêmes possibilités que celles en Java. En interne dans les navigateurs, les « class » sont transformées en prototype comme vu au chapitre 3.6. Si vous faites un « console.dir(Personne) », vous verrez la méthode toString dans « prototype ».

**class Personne {**

**constructor(prenom, nom, age) {  
 this.prenom = prenom;  
 this.nom = nom;  
 this.age = age;**

**}**

**toString() {**

**return this.nom + " " + this.prenom + " (" + this.age + ")";**

**}  
}**



Il est possible de définir des champs et méthodes privés avec le « # », mais cela n'est pas encore bien implémenté dans les différents navigateurs.

### Chalenge (30 à 60 minutes environ)

Si votre exercice 10 fonctionne, copiez-le en *«****ex11-js-poo-class***».

On vous lance ici un défi qui va faire ressembler votre code à du Java en utilisant le concept de *« classe ».* Votre challenge est donc de transformer les modules de code de cet exercice en créant des **classes** pour :

* Personne
* Worker
* Ctrl

Ensuite, la seule grande difficulté est de savoir comment créer un objet contrôleur (ctrl) pour qu’il soit disponible pour la vue ? On vous donne ici la solution : l’objet *« ctrl »* doit être stocké dans l’objet *«****window****»* du navigateur car les méthodes de l’ « indexCtrl » sont appelées depuis l’ « index.html ». L’objet window doit être ajouté à la variable ou on ne met rien et la variable devient globale, surtout pas de let !

/\*

\* DOM PRET : DEMARRAGE DE L'APPLICATION dans le fichier indexCtrl.js

\*/

**document.onreadystatechange = function () {**

**if (document.readyState === "complete") {**

**window.ctrl = new Ctrl(); // ou ctrl = new Ctrl();**

**ctrl.afficherPersonnes();**

**}**

**};**

**Important**: il faut faire référence aux attributs et méthodes de la classe en utilisant le mot-clé **this**. Comme c’est le Ctrl qui crée le Worker, on peut accéder aux attributs en utilisant this.wrk.personnes ou this.wrk.ajouterPersonne(p) pour une méthode.

/\*

\* Première ligne de la classe Ctrl

\*/

**class Ctrl {**

**constructor() {  
 this.wrk = new Worker();  
 }**

Tout cela s'approche de la programmation Java !

## Exercice 12 – les closures – compteur (45’)

Avant de se lancer, voici quelques remarques concernant les déclarations de variables :

**var maVar = 10 ;**

**var maVar = 100 ;**

Cette variable **maVar** est maintenant stockée dans l’objet window (console.log(window) pour le visualiser) qui est l’objet de base de votre page affichée dans le navigateur. Le fait de redéclarer **maVar** ne génère pas d’erreur et écrase la valeur précédente.

**meineVar = 10 ;**

Si vous oubliez le mot « var », le système stocke la variable dans l’objet window (contexte global)

**for (var i=0 ; i<10 ; i++){**

**console.log(i) // 0 à 9**

**}**

**console.log(i) ; // => 10**

La variable **i** est présente dans l’objet window, après la boucle la variable est toujours là.

**function maFct(){**

**for (var j=0 ; j<10 ; j++){**

**console.log(j) // 0 à 9**

**}**

**}**

**maFct() ; // exécution de ma fonction**

**console.log(j) ; // => erreur (j indéfini)**

La variable **j** est maintenant indéfinie, car la déclaration de **j** se fait dans une fonction. Les variables de la fonction sont dans le contexte local de la fonction (scope de la fonction)

**function meineFct(){**

**for (k=0 ; k<10 ; k++){**

**console.log(k) // 0 à 9**

**}**

**}**

**meineFct() ; // exécution de ma fonction**

**console.log(k) ; // => 10**

La variable **k** est à nouveau définie, car la non-utilisation de « var » fait passer la variable dans le contexte global.

Un nouveau mot-clé est « let » est apparu et à moyen terme va remplacer le « var ». Le comportement du « let » est plus proche des variables des langages de programmation. Une variable est définie dans le bloc et dans les sous-blocs

**for (let i=0 ; i<10 ; i++){**

**console.log(i) // 0 à 9**

**}**

**console.log(i) ; // i indéfini**

**FAVORISEZ L’EMPLOI DU LET**

Il reste une spécialité du langage JavaScript qui est la closure (fermeture). Je vous propose de lire le chapitre du document de Pierre Giraud sur les closures. Ensuite on va se baser sur le compteur présent dans le PDF et faire un exercice.

Depuis votre outil de développement, veuillez copier l’exercice *« ex03… »* vers *« ex12-js-closure »*.

Ensuite, réalisez encore le travail suivant :

Supprimer le contenu du fichier *« indexCtrl.js »* pour le remplacer par le compteur du PDF, créer un tableau avec 3 compteurs (3x la fonction dans un tableau) et ajouter 1 seul écouteur avec un index pour les 3 boutons.

Afficher la valeur du compteur directement en gras sur le bouton :

**function actionCompteur(idx) {**

**let val = tab[idx-1].call(); // appelle explicite de la fonction du tableau**

**document.getElementById("button"+idx).innerHTML = "Compteur …";**

**}**

### Théorie

Le plus grand défaut de tous les exercices faits jusqu’à présent (sauf avec les classes) est la *«****portée globale****»* de tout le code produit. Par exemple, une fonction *« execute »* que nous écririons pourrait être écrasée par un autre code d’une bibliothèque avec le même nom de fonction.

Comment ont donc fait les programmeurs de librairies JavaScript comme JQuery, Angular ou VueJS pour ne pas interférer avec nos propres méthodes qui pourraient écraser les leurs (ou le contraire) ?

Et bien, ils ont utilisé le principe des *«****closures****»* (fermetures) et de l’ *« Immediately-Invoked Function Expression »* (**IIFE**) ou « *expression de fonction immédiatement invoquée »*.

Dans ce premier exercice, on va d’abord s’occuper du principe des *« closures »*.

**But de l’exercice ?**

Le but principal de cet exercice est déjà de **reconnaitre** une *« closure »* dans un code JavaScript et de comprendre que celle-ci **mémorise** l’état des variables ou objets dans son champ de vision et ceci AVANT même que celle-ci ne soit exécutée ! En pratique, cela permet surtout de mémoriser du code qui sera exécuté plus tard.

**Comment reconnait-on une closure ?**

Une closure (fermeture) est une **fonction** **anonyme** **dans une fonction** qui rend *«****privé****»* tout le code qui se retrouve à l’intérieur. Les variables locales d'une closure restent en mémoire à contrario d'une méthode normale qui détruit ses variables locales en fin d'exécution. Cela permet donc de protéger ce code vis-à-vis de codes externes qui ne pourront donc l’endommager.

**function bonjour(prenom) {**

**let resultat = "Bonjour " + prenom; // variable qui reste en mémoire**

**let maClosure = () => console.log(resultat); // fonction anonyme avec flèche**

**// let maClosure = function() { // fonction anonyme classique**

**// console.log(resultat);**

**// };**

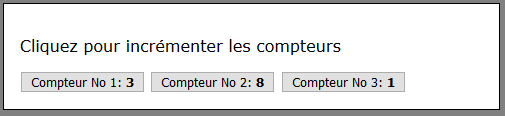
**return maClosure;**

**};**

**let fct = bonjour("Bob");**

**fct();**

### Maquette de la vue



### Code HTML

<body>

    <div id="container">

      <p>Cliquez pour incrémenter les compteurs</p>

      <button id="button1" onclick="actionCompteur(1);">Compteur No 1</button>

      <button id="button2" onclick="actionCompteur(2);">Compteur No 2</button>

      <button id="button3" onclick="actionCompteur(3);">Compteur No 3</button>

    </div>

  </body>

Dans le document de Pierre Giraud, il parle de closure. Veuillez reprendre l'exemple avec le compteur et créer un tableau de 3 fonctions qui seront appelées par les clics sur les boutons.

## Exercice 13 – les IIFE en JavaScript

Une fonction est premièrement définie et ensuite exécutée une à plusieurs fois. Voici deux manières d’écrire une fonction JS :

**// Déclaration d’une fonction**

**function a() {**

**let val = 1;**

**console.log(val) ;**

**}**

**a(); // => exécution de la méthode et affichage de 1 dans la console**

**// si on écrit a(); avant la déclaration de la fonction, cela fonctionne !**

**// Déclaration d’une expression fonction (fonction anonyme)**

**let b = function() {**

**let val = 2;**

**console.log(val) ;**

**} ;**

**b(); // => exécution de la méthode et affichage de 2 dans la console**

**// Déclaration d’une fonction flèche (arrow function) ; 3 cas de figure**

**let c = (a,b) => a + b ;**

**console.log(c(2,5)) ; // = 7**

**let d = val => val\*val ; // avec un paramètre**

**console.log(d(5)); // = 25**

**let e = (a,b) => { // avec plusieurs instructions**

**let somme = a+b;**

**return somme;**

**};**

**console.log(e(5,7)); // = 12**

Il est, des fois, nécessaire de créer une fonction et l’exécuter directement : IIFE *« Immediately-Invoked Function Expression ».* Voici deux manières d’écrire une fonction JS IIFE:

**(function() {**

**let val = 3;**

**console.log(val) ;**

**})() ; // => exécution et affichage de 3 dans la console**

**(() => {**

**let val = 4;**

**console.log(val) ;**

**})() ; // => exécution et affichage de 4 dans la console via une fonction flèche**

Dans la console de votre navigateur préféré, testez les différentes situations si dessus.

Ensuite, transformez toutes ces fonctions (val 1 à val 4 uniquement) en additionneur en ajoutant un seul paramètre et en l'additionnant à *«* val *»*. Cela correspond au travail à effectuer pour cet exercice, les 4 fonctions doivent se retrouver dans votre RP.

~~On va mettre en place une IIFE en partant de notre exercice~~ *~~«~~****~~ex11-js-poo-class~~***~~» que vous allez renommer avec un «~~**~~13~~**~~» et y ajouter «~~**~~iffe~~**~~»~~

~~Le but est d’éliminer le fichier « app.js ». Avant de l’effacer, il faut copier la fonction au sommet du fichier « indexCtrl.js » et faire une IIFE pour que cette fonction soit exécutée au chargement du fichier. Le corps de la fonction sera réellement exécuté lorsque le document est « complete ».~~

Il est aussi possible d’ajouter deux IIFE dans l’exercice « ex09-js-poo» sur le ctrl et le worker. C’est très courant, car les programmeurs ont encore de la peine à passer de la fonction prototype à la classe.

### Paramètre d’une fonction JS

Vous êtes à l’aise avec les fonctions Java, c’est très ressemblant en JS et c’est aussi ressemblant dans beaucoup d’autres langages. Un avantage en JS est de ne pas renseigner tous les paramètres au moment de l’appel. Cette façon de faire est très courante en JS et cela perturbe un peu les développeurs Java !

**function add(a, b, c){**

**return a + b + c;**

**}**

**console.log(add(1,2,10); // => 13**

**console.log(add(1,2)); // => NaN car la variable c n’a pas été initialisée**

**function add(a, b, c=0){ // assignation d’une valeur par défaut de C**

**return a + b + c;**

**}**

**console.log(add(1,2,10)); // => 13**

**console.log(add(1,2)); // => 3**

# Apprentissage de jQuery

Dans le chapitre précédent, nous avons utilisé du JavaScript dans de petits exercices. Nous avons aussi appris à modulariser notre code grâce aux closures et à IIFE (ou avec ES6). Mais qu’en est-il pour créer de vraies applications utiles côté client ?

Il y a au moins 3 problèmes que JavaScript ne résout pas bien :

* On ne peut manipuler qu’on objet du DOM à la fois (donc il faut écrire un nombre de lignes incroyable pour modifier par exemple tout une série de <LI>...</LI>) ;
* Il est extrêmement verbeux (on doit écrire beaucoup pour obtenir peu) ;
* On doit parfois écrire plusieurs codes différents suivant les navigateurs.

JQuery est une bibliothèque JavaScript très utilisée et appréciée dans le monde des programmeurs web. Elle résout la plupart des problèmes énoncés ci-dessus (mais en laisse tout de même quelques-uns en suspens, comme le *« data-binding »*).

Le côté magique de JQuery tient à sa fonction **$()** ou **jQuery()** qui permet d’accéder aux objets du DOM, de gérer des évènements, de produire des effets graphiques ou encore d’implémenter AJAX (requêtes asynchrones en JavaScript vers des serveurs).

## Exercice 14 – bases de jQuery (90’)

L’exercice 14 est en fait déjà … terminé, car il s’agit d’une grosse page HTML composée de diapositives (des DIV) sur jQuery dans lesquelles on peut déjà apprendre et tester un certain nombre de choses. Vous trouvez cette démo avec le lien suivant :

<http://www.jcsinfo.ch/demo/jquery>.

Le but est de lire ATTENTIVEMENT ces diapositives et de TESTER les exemples fournis (souvent il y a un bouton *« run »* pour le faire). Il y a aussi des SITES Internet qui vous sont proposés. Vous devez les avoir consultés au moins une fois et avoir mémorisé ces liens dans un dossier *« 307 »* sur la barre des favoris de Chrome.

### JS vs JQuery

Voici quelques différences pour le traitement de base des éléments du DOM

**let info = document.getElementById("info");**

**//let info = document.querySelector("#info");**

**info.textContent="Une info JavaScript!";**

**info.innerHTML=("<b>C'est bold</b>");**

**info.style.backgroundColor="blue";**

**info.style.color="red";**

**info.addEventListener("click", () => { info.textContent="click JS"});**

**info.append(document.createElement("div"));**

**$("#info").text("C'est une info JQuery");**

**$("#info").html("<b>C'est bold</b>");**

**$("#info").css("color","red");**

**$("#info").css({"color": "red"});**

**$("#info").css({color:"red", backgroundColor:"blue"});**

**$("#info").css({"color":"green", "background-color":"cyan", "font-size":"1.5em"});**

**$("#info").click(() => { $("#info").text("click JQuery")});**

**$("#info").append("<div>C'est un div</div>");**

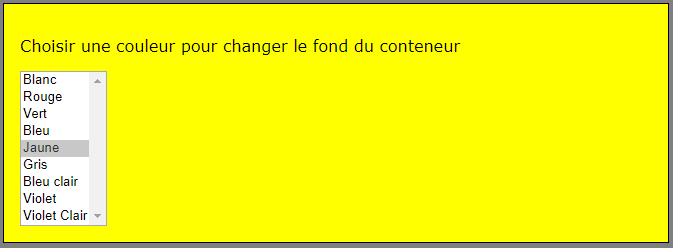
## Exercice 15 – première utilisation de jQuery (90’)

Depuis votre outil de développement, copier *« ex01… »* vers *« ex15-jquery-premiers-pas »*.

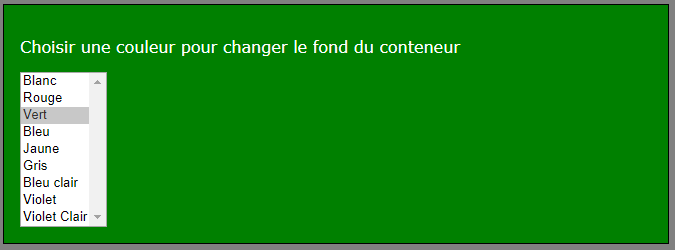
Le but est de créer une vue avec le code HTML ci-dessous, puis de gérer cette vue (changer la couleur du fond du conteneur) avec l’un ou l’autre des 2 contrôleurs proposés (en pur JavaScript ou avec un maximum de JQuery).

### Maquettes

Au démarrage, remarquez le *«****jaune*** *»* sélectionné et la couleur affichée par défaut :



Après sélection du vert, remarquez la couleur du texte affiché qui change du noir au blanc. Cela est valable pour les couleurs *«****vert****»*, *«****bleu****»* et *«****rouge****»* :



### Code HTML

Voici une partie du code HTML. Remarquez la sélection du *«****jaune****»* par défaut et l’appel d’une méthode *«****changerCouleur****»*.

**<div id ="container">**

**<form>**

**<p id="titre">  
 Choisir une couleur pour changer le fond du conteneur**

**</p>**

**<select size="9" id="couleurs"  
 onChange="ctrl.changerCouleur(couleurs.value);">**

**<option value="white">Blanc</option>**

**<option value="red">Rouge</option>**

**<option value="green">Vert</option>**

**<option value="blue">Bleu</option>**

**<option selected value="yellow">Jaune</option>**

**<option value="#F0F0F0">Gris</option>**

**<option value="#0FABCD">Bleu clair</option>**

**<option value="#FB33BB">Violet</option>**

**<option value="#CDB0DC">Violet Clair</option>**

**</select>**

**</form>**

**</div>**

### Les contrôleurs

Le contrôleur à tester sera choisi dans le code HTML (l’un OU l’autre) :

**<title>Ex15 - Première utilisation de JQuery</title>**

**<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>**

**<!-- <script src="js/indexCtrlJS.js"></script> -->**

**<script src="js/indexCtrlJQ.js"></script>**

**</head>**

Implémentez la méthode *«****changerCouleur****»* dans les 2 contrôleurs pour qu’elle effectue le travail requis.

**indexCtrlJS.js**

/\*

\* 1. DOM PRET : DEMARRAGE DE L'APPLICATION

\*/

**document.onreadystatechange = function () {**

**if (document.readyState === "complete") {**

**let couleur = document.getElementById("couleurs").value;**

**changerCouleur(couleur);**

**}**

**};**

**/\***

**\* 2. METHODES PRIVEES**

**\*/**

**function changerCouleur(couleur) {**

**...**

**}**

**indexCtrlJS.js avec une classe Ctrl**

**document.onreadystatechange = function () {**

**if (document.readyState === "complete") {**

**ctrl = new Ctrl();**

**let couleur = document.getElementById("couleurs").value;**

**ctrl.changerCouleur(couleur);**

**}**

**};**

**class Ctrl {**

**constructor(){**

**}**

**changerCouleur(couleur) {**

**…**

**}**

**}**

**indexCtrlJQ.js**

/\*

\* 1. DOM PRET : DEMARRAGE DE L'APPLICATION

\*/

**$(document).ready(function () {**

**let couleur = $("#couleurs").val();**

**changerCouleur(couleur);**

**});**

**indexCtrlJQ.js avec une classe Ctrl**

/\*

\* 1. DOM PRET : DEMARRAGE DE L'APPLICATION AVEC CLASSE CTRL

\*/

**$(document).ready(function () {**

**ctrl = new Ctrl();**

**let couleur = $("#couleurs").val();**

**ctrl.changerCouleur(couleur);**

**});**

JQuery a une panoplie d’effets et d’animations. On peut citer le ***« fade out****»* pour faire disparaitre le conteneur ou ***« fade in****»* pour le faire réapparaitre. La méthode *«****fadeToggle****»* passe d’un état in à un état out ou inversement. La méthode *«****toggleSlide****»* qui fait comme son nom l’indique un effet de slide.

Tentez de faire disparaitre le conteneur avec un effet ***« fade out****»* d’une seconde. Une fois que l’animation est terminée, vous activez le changement de la couleur et la réapparition du conteneur avec un *«****toggleSlide****»* utilisez Internet pour trouver un exemple ou des références sur ces méthodes.

## Exercice 16 – lecture jQuery & Bootstrap (facultatif)

Tout développeur de page web doit tenir compte de différents types de navigateur avec des moteurs de rendu de pages qui ne fonctionnent pas la même manière ou d'autres qui n'implémentent pas les dernières fonctionnalités. Cela a toujours été un casse-tête qui nécessite beaucoup de tests et qui entraine beaucoup de pertes de temps. Les frameworks, type JQuery et d'autres, s'occupent d'une bonne partie de cette problématique. Il reste à gérer la taille des écrans dans un environnement fragmenté entre les desktops, les laptops, les tablettes et les mobiles de toutes tailles. Bootstrap va vous aider à faire des sites "responsive" qui signifient que l'organisation des pages va changer en fonction de la taille de l'écran. Ce framework vous propose de nouveaux composants ainsi que des validations évoluées de formulaire. Il est la propriété de Twitter.

On ne va faire d'exercices sur ce framework, mais on vous laisse du temps pour lire et faire des essais si vous en avez l'envie. Il sera utilisé en 4e année pour développer des sites pour l'école et les entreprises.

Dans le monde web, les frameworks ont des évolutions rapides liées aux évolutions du JavaScript et des frameworks eux-mêmes. Lors d'un changement de version majeure, il est souvent nécessaire de réécrire toute ou partie du front-end car des composants disparaissent et d'autres apparaissent et même les manières de structurer une page change. La version actuelle est la 4.3, mais la version 5 bêta est en cours de validation.

Voici des liens pour vous documenter :

<https://www.pierre-giraud.com/bootstrap-apprendre-cours>

bootstrap-4-premier-pas.pdf (fichier sur le partage)

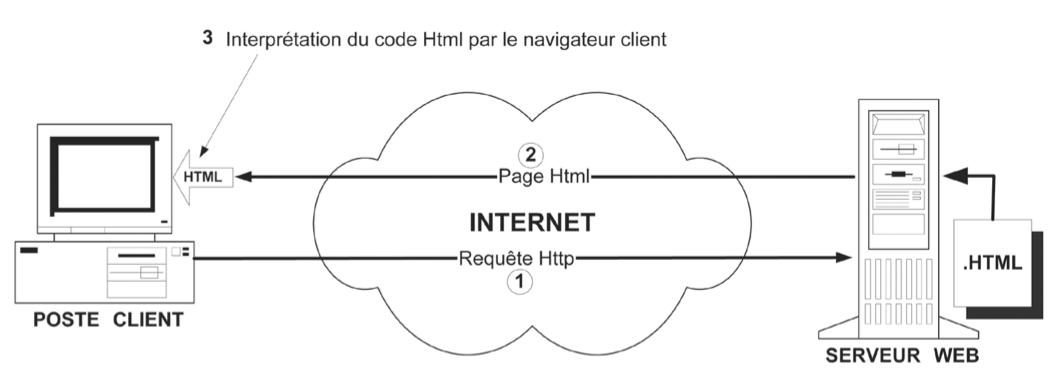
~~Vous documentez les éléments importants dans votre RP (min 1 page, max 2 pages)~~

# Web services

Pour dynamiser votre site web, les animations c'est bien joli, mais c'est mieux avec des informations qui se mettent à jour en fonction des choix de l'opérateur. Un moyen commode de discuter avec un serveur, ce sont les web services.

## Théorie (à lire, très important)

Les échanges entre un navigateur (client) et un serveur web sont basés sur un protocole appelé HTTP (**Hyper Text Transfer Protocol**) ou HTTPS sa version sécurisée. Ce protocole est à l’origine même d’Internet. Ainsi, une URL tapée dans un navigateur fait office de requête HTTP de type GET invite généralement un serveur à retourner une information, le plus souvent une page au format HTML :



Source : livre jQuery-Ajax avec PHP (éditions Eyrolles)

GET est sans effet sur la ressource, il doit être possible de la répéter, à contrario d'une requête POST souvent utilisé pour la soumission des données d'un formulaire dans le but de changer les ressources sur le serveur. Il existe aussi PUT et DELETE. Ces 4 commandes forment les commandes CRUD très connues aussi dans le monde des bases de données.

De nos jours, bon nombre de sites sont devenus de vraies **applications web** (pensez à GABI ou EMF-parking par exemple). Quand vous travaillez avec ces applications, vous ne recevez plus des pages HTML en retour, mais bien des données en **JSON**, **XML**, …, qui sont affichées de manière appropriée : souvent dans un calque de type **<div id="data">…</div>**, mais cela peut aussi être dans une vue *« Google Map »* par exemple.

De nombreuses firmes dans le monde ont d’abord développé des services web pour leurs propres besoins internes qu’ils ont regroupés dans une **API** (Application Programming Interface).

Parfois, ils ont *« externalisé »* certaines fonctionnalités pour les fournir gratuitement ou avec un abonnement payant à :

* Des développeurs indépendants ;
* Des développeurs d’autres firmes.

Une **API** peut être vue comme une alternative à quelque chose que pratiquement tous ceux qui ont utilisé un jour une application connaissent : **l'interface utilisateur**. Mais au lieu d’être destinées à des utilisateurs finaux, les API sont destinées aux développeurs d’application qui pourront programmer leurs propres applications basées sur les services web que fourniront ces API.

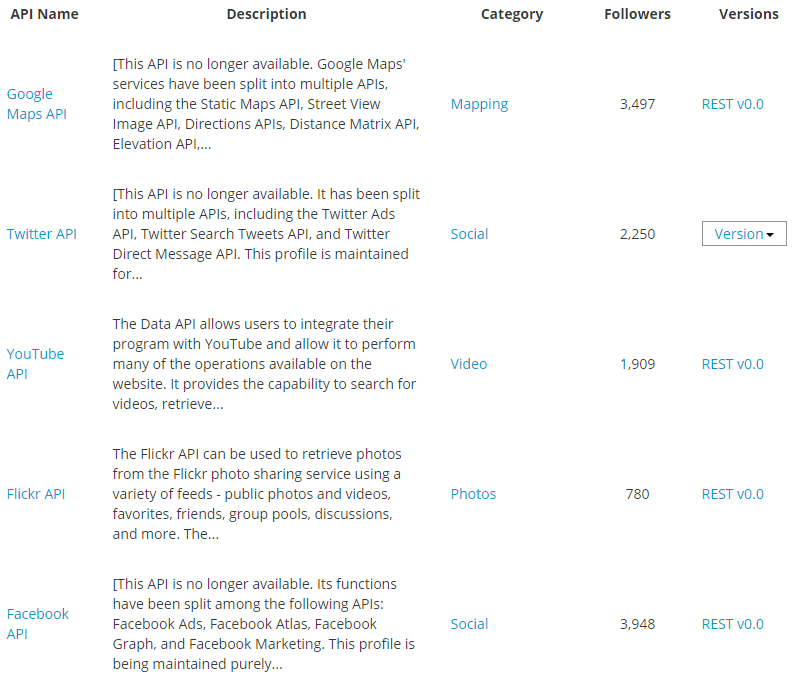
De nos jours, une véritable industrie des API s’est développée offrant (gratuitement ou en payant) de multiples **services** à diverses firmes.

Ainsi le site suivant propose à peu près 23’000 API.

<https://www.programmableweb.com/category/all/apis>

Certains sont gratuits et ouverts, pour d’autres il faudra créer un compte et obtenir une clé d’activation pour chaque requête (parfois gratuite, parfois payante).

Dans le tableau qui suit (issu du lien précédent), on voit bien que ce sont principalement les grosses firmes américaines qui ont commencé avec les *« web API »* :



Il existe 2 types de *« web services »*qui nécessitent une approche différentiée :

* les services web de type *«****REST****» (REpresentational State Transfer)*<https://openclassrooms.com/fr/courses/3449001-utilisez-des-api-rest-dans-vos-projets-web>
* les services web de type *« WS-\* »* basés sur les technologies *«****SOAP****»* (Simple Object Access Protocol) et *«****WSDL****»* (web Services Description Language). Les services de ce type sont encore largement implémentés dans les produits Microsoft.

Dans la suite des exercices de ce document, nous nous attarderons plus particulièrement sur les web services de type REST, beaucoup plus tendances et beaucoup plus faciles à appréhender.

## Exercice 17 – test de web services (60’)

Dans cet exercice, nous allons simplement découvrir un peu de théorie, télécharger un outil gratuit et tester des *« web services »* avec cet outil.

Un outil très utilisé est l'application Postman qui nécessite de s'enregistrer, mais qui reste gratuite. Un autre outil à installer accessible depuis Chrome qui s'appelle Arc via le lien chrome://apps.

### Test d’une API de type REST

Sous <http://www.jcsinfo.ch/parlement/assets/api/index.html>, vous pouvez découvrir une API de type REST (chaque lien retourne une donnée spécifique), développée en JAVA spécialement pour les modules 101 et 307. Cette API permet des requêtes sur des données de *«****conseillers nationaux suisses*** *»* depuis 1868.

On y voit toutes les *« routes »* possibles, appelées aussi **URI** (Unified Resource Identifier) qui représentent les **chemins d’accès uniques** à des données spécifiques. Si on clique sur les liens, on verra les différents retours possibles en HTML, JSON et XML.

Une API fournit généralement une liste des fonctionnalités permises avec des exemples des requêtes. Souvent, les API décrivent également le type de retour avec un exemple. Et parfois, toute une documentation explique en détail le fonctionnement d’une seule URI.

Sous <http://parlement.jcsinfo.ch/>, on peut découvrir cette même API utilisée dans une vraie application web développée en JavaScript, jQuery et Bootstrap. On y découvre que des données retournées peuvent aussi être interprétées et représentées sur une *« Google Map »*.



### Exemple de web service REST en PHP

On pourrait programmer le même web service en PHP et le mettre à disposition sur un site emf-informatique.ch, puisque ces sites supportent Apache et PHP.

Ci-dessous, deux web services PHP que l'on va utiliser tout prochainement. Le premier est de type POST retournant du XML. Le second fait la même chose mais est de type GET retournant du JSON. C'est normalement le second qu'il faut favoriser pour deux raisons. Comme l'appel du web service ne modifie pas la base de données et que l'on peut l'appeler aussi souvent que l'on veut, on doit utiliser GET et la réponse JSON (**J**ava**S**cript **O**bject **N**otation) est plus simple à traiter en JavaScript.

**<?PHP**

**// convert\_temp\_p\_xml.php**

**// WebService Conversion de température POST et réponse XML**

**// Naël Telfser**

**//**

**header('Content-Type: application/xml');**

**$temp = intval($\_POST['Temperature'], 10);**

**$from = $\_POST['FromUnit'];**

**$to = $\_POST['ToUnit'];**

**​**

**if($from === 'C'){**

**$tempf = $temp\*9/5+32;**

**} else if($from === 'F') {**

**$tempf = ($temp-32)\*5/9;**

**}**

**$result = "<temperature>" . $tempf . "</temperature>";**

**echo $result;**

**?>**

**--------------------------------------------------------------------------------------**

**<?php**

**// convert\_temp\_g\_json.php**

**// WebService Conversion de température GET et réponse JSON**

**//**

**header('Content-Type: application/json');**

**$temp = intval($\_GET['Temperature'], 10);**

**$from = $\_GET['FromUnit'];**

**$to = $\_GET['ToUnit'];**

**if($from === 'C'){**

**$tempf = $temp\*9/5+32;**

**} else if($from === 'F') {**

**$tempf = ($temp-32)\*5/9;**

**}**

**$result = array('temperature' => $tempf);**

**echo json\_encode($result); // fonction php pour transformer des données au format JSON**

**?>**

### Outils de test

Certes on pourrait utiliser un navigateur tel que Chrome, mais nous verrons que les navigateurs possèdent une sécurité qui va beaucoup nous gêner à un moment donné : l’interdiction de faire des appels *« cross domain »*.

En effet, si on se retrouve à faire une requête HTTP depuis *« localhost »* ou *« emf-informatique.ch »* vers un service web sur *« www.w3schools.com »*, tous les navigateurs vous indiqueront une erreur d’accès *«****cross-domain****»* pour des raisons de sécurité évidente : certains services cachés dans le code du développeur pourraient envoyer à votre insu, par exemple vos coordonnées bancaires vers une personne ou une organisation malveillante.

Note pour Chrome :

On peut télécharger un plugin *« CORS »* qui peut beaucoup aider pour ce problème :

|  |  |
| --- | --- |
|  | Une fois ce plugin actif, un clic sur l’outil présenté à droite permet d’inhiber le test *« cross-domain ».* |
|  |  |

### Si vous avez la main sur le serveur, vous pouvez autoriser le cross-domain dans la réponse du serveur en modifiant le header avec l'instruction PHP suivante: header("Access-Control-Allow-Origin: \*");

### Test du service web de conversion de températures

Pour tester les services web de température créés en PHP, nous devons fournir les **4 informations** suivantes pour le POST, mais uniquement 2 pour le GET avec l'URL qui contient l'adresse ainsi que les data à transmettre :

**URL** : <https://monnom.emf-informatique.ch/307/convert_temp_p_xml.php>

**Method**: POST

**Content-type**: application/x-www-form-urlencoded le plus commun

**Data**: Temperature=20&FromUnit=C&ToUnit=F les données

**URL** : <https://monnom.emf-informatique.ch/307/convert_temp_g_json.php?Temperature=20&FromUnit=C&ToUnit=F>

**Method**: GET

Pour tester la version PHP, veuillez créer le fichier PHP avec le code fourni page précédente et le stocker sur votre site personnel par exemple dans « 307/convert-temp\_p\_xml.PHP » « 307/convert-temp\_g\_json.PHP »

## Exercice 18 – application HTML5 d’appel à un web service (90’)

Vous voilà enfin prêt pour effectuer de vraies requêtes entre un client est un serveur, tout en maitrisant la structure MVC d’une l’application avec jQuery

### Théorie

Nous avions déjà abordé la question de la transmission d’informations vers un serveur dans l’exercice 3 (chapitre 2.3). Nous avions alors vaguement parlé de la façon d’envoyer un formulaire via une action spécifiée dans la balise *« form »* avec l’une ou l’autre méthode GET ou POST d’HTTP :

**<form class="formulaire" action="http://localhost/login.php" method="POST">**

**...  
</form>**

Au lieu d’envoyer un formulaire par la propriété *« action »* de la balise *« form »*, nous pouvons également utiliser l’objet *«****XMLHttpRequest****»* ou bien *«* **API Fetch** *» (nouvelle manière de faire)* créés automatiquement par le navigateur. Ces objets sont à la base de la programmation **AJAX** (Asynchronous Java-Script and XML). AJAX n’est rien d’autre que l’utilisation d’HTML5 et du DOM pour effectuer des requêtes HTTP de type *« asynchrone »* vers un serveur, c’est-à-dire des requêtes où le résultat attendu n’est pas immédiatement retourné au client.

Au lieu d’utiliser directement cet objet en JavaScript, nous allons profiter de jQuery et de sa méthode principale : **$.ajax(url, …)**.

### Maquettes

 <div id="container">

     <h3> Convertisseur de température </h3>

     <div class="form-group">

              <label for="celsius1">°C:</label>

              <input type="text" size="20" id="celsius1" name="celsius1" placeholder="une température °C" autofocus

                  onkeyup="ctrl.celsius2Fahrenheit();">

              <label for="fahrenheit1">°F:</label>

              <input type="text" size="20" id="fahrenheit1" name="fahrenheit1" disabled="disabled">

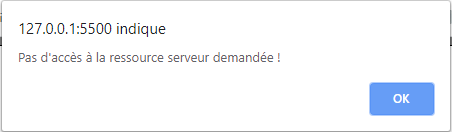
      </div>

</div>





S’il y a le moindre problème dans l’appel AJAX (par exemple l’URL fausse) :



### À faire

Depuis votre outil de développement, veuillez :

* Copier l’exercice « ex01… » vers « ex18-ws-simple ;
* Dans *« /js »*, créer le fichier *« indexCtrl.js », ainsi que « httpServ.js »* qui va faire office de couche de service ; *« index.html »*, *« indexCtrl.js », ainsi que « httpServ.js »* constituent les modules MVC de notre application;
* Ne pas oublier de charger la librairie JQuery dans *« index.html »*;
* Créer la vue;
* Créer les variables *« ctrl et httpServ »* dans *« indexCtrl.js »* pour créer les objets à partir des classes;

**// Attend la fin du chargement de la page**

**$().ready(function () {**

**ctrl = new Ctrl();**

**httpServ = new HttpServ();**

**});**

**class Ctrl {**

**constructor() {**

**}**

**celsius2Fahrenheit() {**

**// Lire les degrés du formulaire**

**let degres = …**

**httpServ.celcius2Fahrenheit(degres,this.OKCelsius2Fahrenheit, this.KOCelsius2Fahrenheit);**

**}**

**KOCelsius2Fahrenheit(xhr) {**

**let erreur = xhr.status + ': ' + xhr.statusText**

**alert('Erreur - ' + erreur);**

**}**

**OKCelsius2Fahrenheit(data) {**

**let temp = $(data).find("temperature").text();**

**let affiche = temp !== "NaN" ? temp : "";**

**// Afficher les degrés dans le formulaire**

**…**

**}**

**}**

​

* créer un HttpServ sous forme de classe

**class HttpServ  {**

**constructor(){**

**}**

**celcius2Fahrenheit(degres, successCallback, errorCallback) {**

**let url = "http://mettrauxpa.emf-informatique.ch/convert\_temp\_p\_xml.php";**

**let param = "Temperature=" + degres + "&FromUnit=C&ToUnit=F";**

***// envoi de la requête***

**$.ajax(url, {**

**type: "POST",**

**contentType: "application/x-www-form-urlencoded; charset=UTF-8",**

**data: param,**

**success: successCallback,**

**error: errorCallback**

**});**

**}**

**}**

​

* copier et tester votre exercice sur emf-informatique avec une URL du genre :  
  http://monnom.emf-informatique/307/ex18

### Amélioration

Si vous mettez une fausse URL, vous avez une erreur pas très parlante (0: error) ou (404: Not Found) pour une page pas trouvée. On va donc faire parler ces erreurs! JQuery possède une série de fonctions autour d'AJAX qui permet beaucoup de combinaisons, je vous laisse les découvrir dans la très riche documentation. Ce que l'on va mettre en place c'est un $.ajaxSetup qui permet de définir des instructions une fois et les requêtes suivantes utiliserons les mêmes instructions, cela permet d'écrire certaines choses qu'une seule fois. On va donc gérer les messages d'erreurs via ce canal:

**// Ajouter ceci, ôter le errorCallback dans le ajax et son paramètre.**

**// Dans l'indexCtrl, appelle cette méthode avec une fonction callback**

**centraliserErreurHttp(httpErrorCallbackFn) {**

**$.ajaxSetup({**

**error: function (xhr, exception) {**

**let msg;**

**if (xhr.status === 0) {**

**msg = "Pas d'accès à la ressource serveur demandée !";**

**} else if (xhr.status === 404) {**

**msg = "Page demandée non trouvée [404] !";**

**} else if (xhr.status === 500) {**

**msg = "Erreur interne sur le serveur [500] !";**

**} else if (exception === 'parsererror') {**

**msg = "Erreur de parcours dans le JSON !";**

**} else if (exception === 'timeout') {**

**msg = "Erreur de délai dépassé [Time out] !";**

**} else if (exception === 'abort') {**

**msg = "Requête Ajax stoppée !";**

**} else {**

**msg = "Erreur inconnue : \n" + xhr.responseText;**

**}**

**httpErrorCallbackFn(msg);**

**}**

**});**

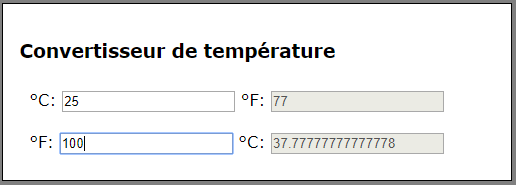
**}**

## Exercice 19 – application HTML5 avec double service web (45’)

On va donc convertir aussi des degrés Fahrenheit en degrés Celsius, mais en utilisant le GET et recevoir du JSON

Depuis votre outil de développement, on vous demande de copier l’exercice précédent vers *« ex19-ws-double-jquery.js »* et de modifier la vue comme le montrent les maquettes suivantes :

### Maquettes



### Marche à suivre

La nouvelle conversion en Celsius est très semblable, il faut :

* Dupliquer les éléments du formulaire et renommer les textes, noms et id
* Dupliquer les 2 méthodes de ctrl tout en sachant que l'on reçoit un objet JSON
* Dupliquer la méthode du HttpServ, configurer la requête en GET et adapter l'URL et le param.
* Tester

## Exercice 20 – application *« login »* sur un serveur PHP (120’)

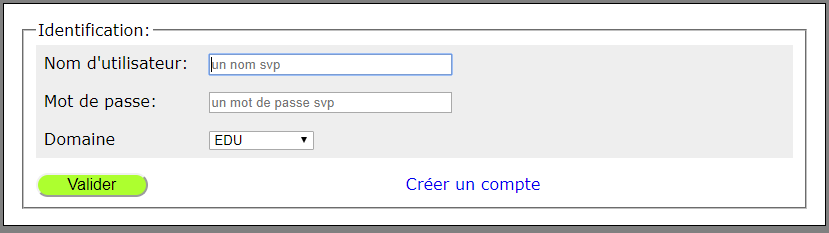
L’exercice 20 est basé sur l'exercice 3 mais il faut récupérer le projet sur le partage car on va y trouver une structure avec plusieurs fichiers.

On va découper l'application en vue, chaque vue ayant son contrôleur. Les vues vont être chargées dynamiquement. L'URL ne va pas changer à la méthode d'une application *«* SPA - Single Page Application *».*

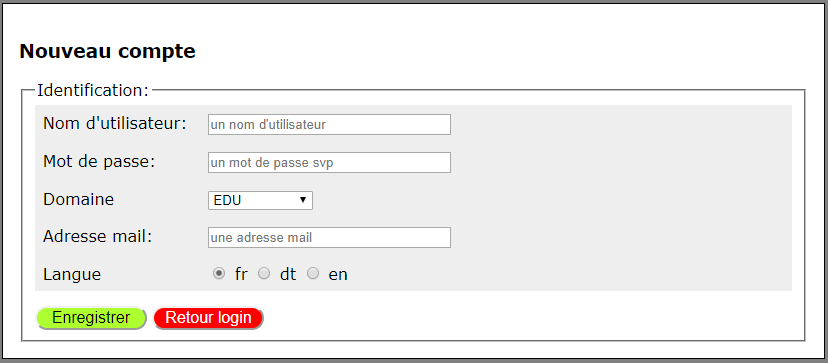
*J'ai créé 3 logins en dur sur le site mettrauxpa. Je vous donne le script PHP pour connaitre les arguments qu'il faudra transmettre en POST. Il retourne la langue "en, de ou fr" si cela se passe bien, sinon KO.*

### Vues modifiées

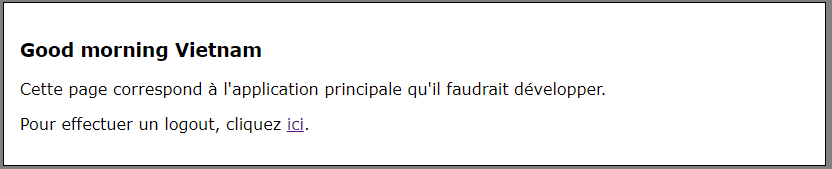
#### Login avec choix du *« domaine »* en plus



#### Création de comptes avec *« domaine »* en plus et sans confirmation *« mdp »*



#### Page après login pour Doe (en anglais)



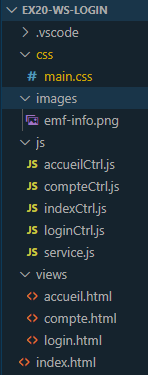
### Comment a été transformée l’application ?

J'ai trois pages qui sont charger à tour de rôle dans un div de la page principale "index.html" via une fonction JQuery "load". Chaque page a son contrôleur. Le "service.js", c'est l'ancien httpServ avec, en plus, la méthode de chargement des vues.

Deux objets "indexCtrl" et "service" sont globaux donc atteignables depuis tous les fichiers JS. Lors du chargement des vues, on crée le contrôleur associé et on rajoute par programmation les écouteurs nécessaires pour le fonctionnement de la page. La bonne vieille méthode de mettre l'écouteur en dure dans le code html nécessite d'ajouter la référence sur l'objet devant la fonction (écouteur), comme cette référence n'est pas globale, cela génère une erreur et le mal-fonctionnement de l'appli.

Je vous transmets le projet terminé à 95%, il reste deux grosses lacunes et un bug :

* La transmission des données au web service ne se fait pas.
* La page de création de compte ne se charge pas.
* Les données de création de compte transmises au web service sont incorrectes.



## Exercice 21 – application *« streetmap »*

Veuillez récupérer l’exercice présent sur le partage, le charger dans Visual Studio et le tester. C’est déjà fini !

Dans le projet que vous avez déjà commencé, j’espère, il faudra faire apparaitre une page avec une carte. Basez-vous sur cet exercice pour arriver simplement au résultat.