Tutoriel
Réalisation d'un CMAV (*angl* : SCRUD) en HTML/JavaScript

CMAV : Interface de Création Modification Ajout Visualisation des données d'une base.
Anglais : SCRUD : search, create, read, update, delete.

Cet outil relève des bibliothèques indispensables pour une gestion de base de données.

L'idée : Utiliser au maximum le JavaScript, en mode « POO », et Ajax pour l'accès aux données. Le PHP servira seulement à gérer les requêtes SQL sur la base de données.

Mots clés : CMAV, CRUD, SCRUD, HTML HTML5 CSS CSS3 sélecteur FLEX DOM2 PHP PDO Ajax classe class constructor Arrow flèches this contexte iterator

---

Pré-requis pour ce tuto : Bases du HTML / CSS / Langages objets / PHP

Pour prendre immédiatement les bonnes habitudes, nous allons séparer le HTML, le CSS, le Javascript, et le PHP.

Ce tuto explique certaines techniques utilisées.
Les fichiers complets sont joints à ce document.
Une démo est disponible ici.

Sommaire

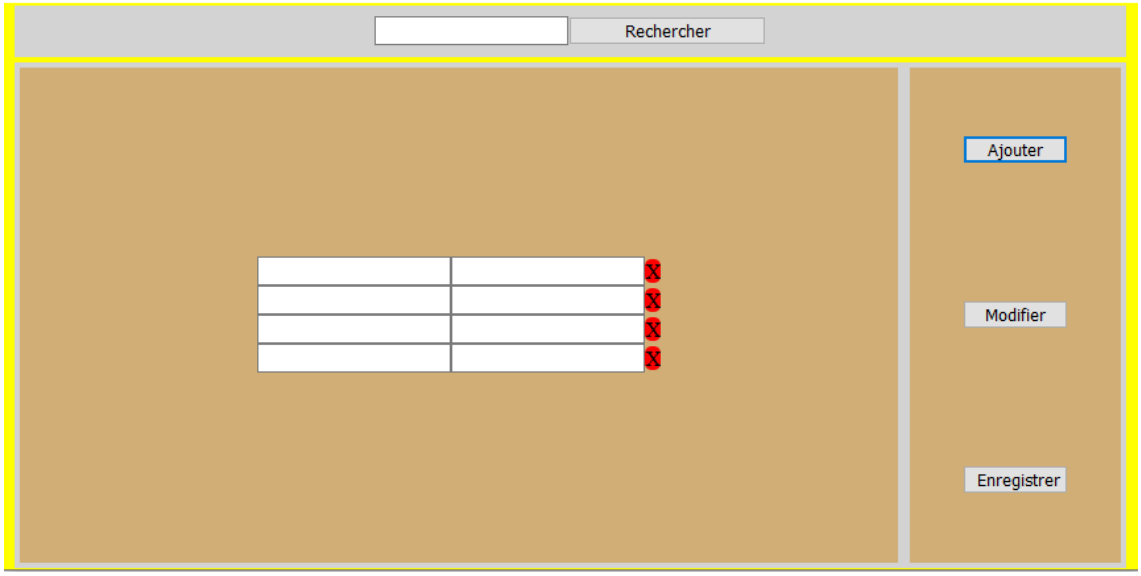
1 Choix du FLEX pour l'interface homme-machine (IHM, angl : UI)

L'IHM s'appuie sur le positionnement FLEX des éléments HTML.
En CSS, le mode FLEX (*display :flex*) s'oppose à la philosophie GRID (*position :absolute*) qui impose une position précise pour chaque élément HTML. Le FLEX surpasse efficacement le réglage CSS « *float* ».

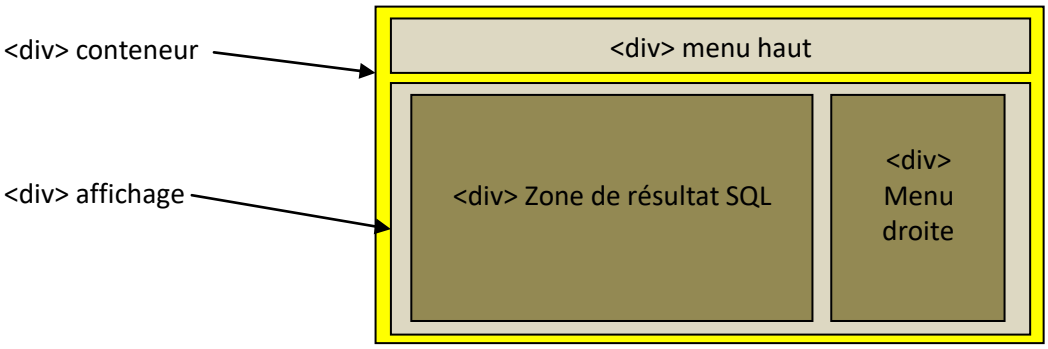
En mode FLEX, on choisit pour chaque <DIV> parente (*parentNode*) la façon dont les éléments enfants (*childNodes*) sont alignés : en ligne, en colonne, centrés, justifiés, …

Un bon tuto se trouve sur l'excellent site ***alsacreation*** :
https://www.alsacreations.com/tuto/lire/1493-css3-flexbox-layout-module.html

On vise ce résultat extrêmement moche mais vous saurez certainement exercer vos talents artistiques en modifiant le CSS. Les couleurs sont là pour montrer la délimitation des :



Détail de la construction : L'utilisation du mode FLEX nous amène à utiliser plus de <DIV> que d'habitude !



Donc en tout 5 pour seulement 3 utilisées. Pas d'inquiétude, c'est normal !! Et ce sera pas + cher…
Les espaces entre les permettent de bien visualiser leur position dans la hiérarchie parent / enfants.

On arrive à cette description HTML :

```
<div class="enColonne contener">
      <div class="enLigne menuHaut"> </div>
      <div class="enLigne affichage">
            <div id='zsql' class="enColonne sql"> </div>
            <div class="enColonne menuDroite"> </div>
      </div>
</div>
```

Voici le CSS qui correspond : Seul le « contener » est positionné en mode absolu. Il faut bien commencer quelque-part…

```
div {
      display:flex;
      align-items:center;
      justify-content:space-around;
}

.enLigne { flex-direction:row; }
.enColonne { flex-direction:column;}

.contener {
      position:absolute;
      top:10%;
      left:0;
      height:90%;
      width:100%;
      background-color:yellow;
}

.menuHaut {
      height:9%;
      width:98%;
      justify-content:center;
      background-color:lightGrey;
}
```

```
.affichage {
      height:89%;
      width:98%;
      background-color:lightGrey;
}

.menuDroite {
      height:98%;
      width:19%;
      background-color:#d0ae76;
}

.sql {
      justify-content:center;
      height:98%;
      width:79%;
      background-color:#d0ae76;
}
```

Je vous laisse réfléchir à l'action des propriétés *flex-direction, justify-content*, et *align-items* …

Il suffit maintenant d'ajouter les balises <input> et <button>
Il est possible de les ajouter par JavaScript mais pour l'instant cela compliquerait le code … et avant, on a des sujets délicats à aborder.
Le code intégral des fichiers HTML et CSS est donné en annexe à la fin de ce document.

2 Où l'on s'attaque au JavaScript : bonjour « THIS » !

Comme on le voit en bas du HTML complet en annexe, on ajoute 2 liens vers les fichiers scripts JS :

- La bibliothèque contenant la classe *cmav*.
- Le programme principal

Rappel sur la notion d'objet et de classe en POO :

Comparons avec les constructions de maisons :

On peut construire tout un lotissement de maisons identiques avec un seul plan.
Le plan prévoit l'emplacement des gens, et des fonctionnalités : Cuisine, SdB, WC, chambre, …
Une fois construite, chaque maison reçoit ses propres occupants, qui utiliseront les fonctionnalités de la maison.

La **classe** correspond au **plan** de la maison.
L'objet correspond à la **maison** construite.
Chaque objet est indépendant et contient ses propres données (les gens, appelés « **propriétés** »), lesquelles utiliseront les fonctionnalités de la classe (les **méthodes**).

Fichier « cl_cmav.js » : Création de la classe *cmav* :

```
class cmav {
      constructor (cible){
            this.zSQL = cible;
      }
}
```

Fichier « principal.js » : Création le l'objet de classe *cmav* :

```
let c = new cmav('#zsql');
```

La méthode « *constructor* » est exécutée à la création de l'objet. Ce nom est imposé.
Ici, on reçoit un argument du programme principal : l'attribut ID de la balise qui recevra le résultat des requêtes SQL.

On crée ensuite une propriété interne à la classe : une variable nommée zSQL. Le préfixe *this* sert à préciser que cette variable appartient au futur objet, et ne sera disponible que dans celui-ci. ***This*** représente le futur objet.

This existe tout le temps, même en dehors d'un objet, et représente le contexte dans lequel s'exécute le *thread* d'exécution. Il faut donc toujours avoir conscience du contexte d'exécution pour ne pas se tromper de ***this.*** Un petit tour de `console.log(this)` permet souvent de lever l'ambigüité.

3 Un premier bouton : Ajouter une ligne vide

J'impose ici un choix stratégique : Utiliser 2 fonctions (méthodes) distinctes pour créer une ligne :

- Une fonction qui crée la ligne
- Une fonction qui place le contenu de la ligne

Nous utiliserons ces fonctions dans le cas d'une nouvelle ligne (Ajouter), de la ligne de titre, mais aussi lorsqu'on affichera les lignes venant de la base de données.

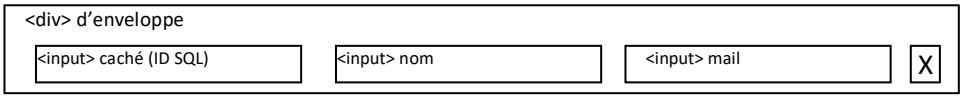
Modification du Fichier « cl_cmav.js »

```
class cmav {
   constructor (cible){
         this.zSQL = cible;
         this.titre = "['xx', 'NOM', 'E-MAIL']" ;
   }

       ajouter () {

       }

       creerLigne(data) {

       }
}
```

Commentaire à propos de la ligne : `this.titre = "['xx', 'NOM', 'E-MAIL']" ;`
On crée une nouvelle propriété pour la classe : un variable qui contient le nom des colonnes affichées.
Le contenu «xx » est un choix arbitraire pour repérer cette ligne et ne pas la confondre avec les lignes de données.

3.1 Modèle d'une ligne

Dans notre exemple, chaque ligne est composée de 2 <input> visibles (nom, adresse mail), d'1 <input> caché (le numéro ID de la table SQL, pour repérer la ligne précisément en cas de modification / suppression), et une pour le pictogramme de suppression.



3.2 La méthode creerLigne (version simplifiée) :

Compléter la méthode « creerLigne » dans le fichier Fichier « cl_cmav.js »

```
creerLigne(data) {
                // Enveloppe de ligne
                var ligne = document.createElement('div');
                ligne.setAttribute('style', 'display:flex; flex-direction:row');

                // Contenu de la ligne
                for (let i = 0; i < data.length; i++) {
                        let unInput = document.createElement('input');
                        unInput.value = data[i];
                        if ( i == 0 ) unInput.setAttribute("type", "hidden"); // Colonne id
                        if ( data[0] != '+') unInput.readOnly  = true; //Pas une nouvelle ligne
                        ligne.appendChild(unInput);
                }

                let suppr = document.createElement('div');
                suppr.innerHTML = 'X';
                suppr.setAttribute('class', 'pictoAnnuler');
                ligne.appendChild(suppr);

                return ligne;
        }
```

La variable « data » contiendra un tableau (array) avec les données d'une seule ligne à placer dans les <input> de la ligne.

3.3 La méthode « ajouter » une ligne vide

Pour ajouter une ligne vide : On utilise un tableau « vide », ou presque … On placera un ID bidon, par exemple « + » pour signaler que c'est une nouvelle ligne, à ne pas confondre avec une ligne qui vient de la base de données et qui possède un ID que l'on veut conserver.

Compléter la méthode « creerLigne » dans le fichier Fichier « cl_cmav.js »

```
ajouter() {
                var cible = document.querySelector(this.zSQL);
                var ligneVide = [];
                for (let i = 0; i < this.titres.length; i++) => { // Même nombre de colonnes que le titre
                        if (i == 0) ligneVide.push('+'); // Symbole pour reconnaitre une nouvelle ligne
                        else ligneVide.push('');
                });
                cible.appendChild( this.creerLigne(ligneVide) );
        }
```

3.4 Bilan

Pour créer des balises dynamiquement avec le JavaScript, on utilise des méthodes du DOM (Document Object Model) :
L'objet ***document*** connait les méthodes *createElement* et *appendChild* qui permettent respectivement de créer une balise et de l'ajouter à la page HTML (en indiquant le nom de la balise de destination). Chaque objet du DOM possède les mêmes méthodes.

Chaque propriété des objets représentant des balises HTML est accessible par le JavaScript soit directement (*value*, *innerHTML*, *id*, …), soit par une méthode de type *setAttribute*.
On note aussi la propriété *length* des variables de type tableau / liste, utilisée dans la boucle FOR.

3.5 On teste ?

Il suffit d'appeler la méthode « ajouter » dans le Fichier «principal.js » :

```
let c = new cmav('#zsql');
c.ajouter();
```

Testez : Une ligne devrait apparaitre dans la « zSQL ».
Pour relier cette action au bouton « Ajouter », on va associer un évènement au clic. Le code final serait donc pour l'instant (dans le Fichier «principal.js ») :

```
let c = new cmav('#zsql');

document.querySelector('#btnAjouter').addEventListener('click', c.ajouter);
```

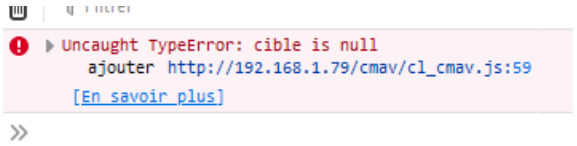
Testez !

Il n'y a pas la ligne de titre … Normal !! Il faut ajouter le code suivant, soit dans le constructeur, soit dans une méthode séparée appelée par le constructeur de la classe (ou par le programme principal) :

```
let cible = document.querySelector(this.zSQL);
cible.appendChild( this.creerLigne(this.titres) )
```

3.6 Catastrophe ! Ca ne marche pas …

Quelle déception ! Le message suivant apparait dans la console :



Du coup, c'est moi qui me sens *null* …. ☹

Analyse : Le thread d'exécution va bien dans la méthode « ajouter » mais cette ligne lui pose problème :

```
var cible = document.querySelector(this.zSQL);
```

Et croyez-moi sur parole : le problème vient de *this*.
L'appel direct d'une fonction (c.ajouter) provoque un changement de contexte : comme la fonction est appelée par le gestionnaire d'évènements lors du clic sur le bouton, le nouveau contexte est l'objet DOM <button>, donc évidemment, pas de propriété « zSQL » dans ce bouton …

ALORS ???? JavaScript ne sait pas faire de la POO proprement ? Parce qu'en utilisant des *function* à la place des classes, ou même en utilisant un bon vieux *« onclick »* de grand-papa, ça marche …
ON NOUS AURAIT DONC MENTI ?????

NON bien sûr : Quand à être moderne, soyons-le jusqu'au bout.
Il existe une nouvelle façon d'appeler une fonction sans créer un nouveau contexte :

3.7 Une fonction ARROW pour sauver le coup !

Le code suivant fonctionne (Yess !) :

```
let c = new cmav('#zsql');

document.querySelector('#btnAjouter').addEventListener('click', () => c.ajouter());
```

La fonction « arrow » (repérable avec son signe =>) ne sert pas simplement à avoir une écriture plus concise des fonctions « inline » anonymes. Elle apporte une vraie révolution : elle ne génère pas un nouveau contexte, donc l'objet *this* reste celui du contexte parent. C'est évidemment incontournable dans le développement POO en JavaScript, vue l'importance de *this* à l'intérieur de la classe.

La syntaxe générale d'une fonction « arrow » anonyme est la suivante :

```
(argument) => { instruction1 ; instrution2(argument) ; … ;}
```

S'il n'y a qu'une seule instruction, on peut se passer des accolades et du point-virgule.

Comme une fonction anonyme est généralement appelée par un ***gestionnaire d'évènement***, ou une méthode de type « ***iterator*** », le ou les **arguments** sont générés automatiquement par le gestionnaire appelant.

Par exemple, utilisation de l'objet « évènement » généré automatiquement par « addEventListener » :
En cliquant, on ajoute 1 au nombre contenu dans un élément <input> nommé « objet » :

```
objet.addEventListener('click', (e) => {
            let a = e.target.value ;
            a++ ;
            e.target.value = a ;
  });
```

Pour tenter de vous impressionner, je vais convertir une portion de code de la méthode *creerLigne* en utilisant l'*iterator* ForEach (méthode intégrée à un objet de type liste) :

Version classique :

```
// Contenu de la ligne
for (let i = 0; i < data.length; i++) {
       let unInput = document.createElement('input');
       unInput.value = data[i];
       if ( i == 0 ) unInput.setAttribute("type", "hidden"); // Colonne id
       if ( data[0] != '+') unInput.readOnly  = true; //Pas une nouvelle ligne
       ligne.appendChild(unInput);
}
```

Version arrow :

```
data.forEach( (valeur, index) => {
       let unInput = document.createElement('input');
       unInput.value = data[valeur];
       if ( index == 0 ) unInput.setAttribute("type", "hidden");
       if ( data[0] != '+') unInput.readOnly  = true;
       ligne.appendChild(unInput);
       });
```

Bon d'accord, on a le même nombre de ligne, mais ça fait quand même du bien ☺

4 AJAX pour interroger la base de données

AJAX sert à appeler une page du serveur sans recharger la page actuelle. Les données renvoyées par le serveur seront reçues en « arrière plan » dans une variable, et un évènement déclenchera le traitement de ces données. Les données reçues sont du texte, structurée ou non. Parmi les formats structurés classiques : csv, xml, json…

Le Json est un choix pratique, bien intégré au JavaScript et au PHP :

- Le PHP reçoit un tableau PHP contenant les données SQL.
- L'objet « tableau PHP » n'est pas utilisable directement par le JavaScript.
- Le PHP convertit le tableau en données structurées JSON,
- Le JavaScript reçoit ces données structurées,
- Le JavaScript convertit le JSON en tableau JavaScript, et donc on sait quoi en faire…

A propos … AJAX = ***A**synchonous **JA**vascript for **X**ml*
Donc créé à l'origine pour les échanges XML … Mais détourné pour être utilisé avec d'autres formats.

4.1 Côté PHP (code simplifié)

Le code suivant fonctionne (à condition d'avoir une base de données sur le serveur...)
Pour notre exemple, il sera dans un fichier nommé « phpajax.php »

```php
// Base de donnée 'cmav', une seule table 'annuaire' :
//                                id (Primary key auto-increment), nom, mail
//
// Connexion à la base de données
	try {
		$pdo = new PDO('mysql:host=localhost;dbname=cmav', 'xxxx', 'xxxxx');
	}
	catch (Exception $e) {
		echo "ERR Exception : ". $e->getMessage();
		die();
	}

// Liste de l'annuaire
if ( $_GET["action"] == "liste" ) {
	$texte = "%".$_GET["q"]."%";
	$req = $pdo->prepare('SELECT * FROM annuaire WHERE nom LIKE ? OR mail LIKE ?');
	$req->execute( array($texte, $texte) );
	$resul = $req->fetchAll(PDO::FETCH_NUM); // Retour à index numérique
	if ( ! $resul )
		echo "KO";
	else
		echo json_encode($resul);
	die();
}
```

Bon, ce n'est pas un tuto PHP …
Pour faire court, on utilise un objet PDO (*Php Data Object*) pour ouvrir un accès à la base de données, et on interroge la base avec la méthode des « requêtes préparées ».
Les données sont récupérées sous la forme d'un tableau à index numériques (PDO ::FETCH_NUM)
Et envoyées au format JSON grâce à la ligne :

```php
json_encode($resul);
```

Simple, non ?

4.2 Côté JavaScript

De ce côté, un échange AJAX commence par la création d'un objet **XMLHttpRequest**.
Ensuite, il faut utiliser les méthodes OPEN, ONLOAD, et SEND de l'objet, et sa propriété RESPONSETEXT

- OPEN : Le site à contacter (l'URL)
- ONLOAD : Le nom de la fonction Callback à appeler quand les données arriveront. Ce peut être aussi une fonction anonyme, classique ou arrow. C'est là qu'on exploitera les données contenues dans la propriété RESPONSETEXT.
- SEND : Déclencher l'échange Ajax. Indispensable !!!

Voilà le code de 2 méthodes à ajouter à notre classe cl_cmav (maintenant vous savez ce que ça veut dire ☺) :

Méthode appelée par ONLOAD :

```
remplir(reponse) { // Afficher les lignes à partir des données reçues
		var cible = document.querySelector(this.zSQL);
		cible.innerHTML = "";

		cible.appendChild(this.creerLigne(this.titres));
		reponse.forEach( (ligne) => cible.appendChild( this.creerLigne(ligne) ));
	}
```

Et la méthode qui déclenche l'Ajax :

```
rechercher() {      //Recherche des lignes correspondant à ce qui est tapé
              // Ce qui est tapé dans la zone de saisie ( id='#zRecherche') :
              var texte = document.querySelector("zRecherche").value;

              // Envoi AJAX de la requête
              var xhr = new XMLHttpRequest();
              xhr.open("GET", "phpajax.php?action=liste&q=" + texte, true);
              xhr.onload = () => {
                            var j = xhr.responseText;
                            if ( j == "KO" )
                                   var tabAjax = []; // pas de profs absents
                            else
                                   var tabAjax = JSON.parse(j);

                            this.remplir(tabAjax);
              }

              xhr.send(); // Go!
       }
```

On remarque la valeur de ONLOAD : une fonction anonyme « arrow » qui en fin appelle la méthode « remplir ».

Côté OPEN, *l'url* appelée contient aussi 2 variables GET (C'est le moment de réviser vos cours sur les formulaires HTML ☹).

Par exemple, si on a tapé toto dans la zone de recherche le mot « toto » :

phpajax.php?action=liste&q=toto

Pas de http, ni de www … puisque la page PHP est sur le serveur, dans le même dossier que le fichier JS.

La variable « action » est utilisée dans le PHP pour choisir la partie PHP à exécuter. C'est une astuce pour n'avoir qu'une seule page PHP pour les différents échanges AJAX.

La variable « q » contient le texte cherché.

Enfin le Graal : L'objet JSON (bibliothèque du JavaScript), grâce à sa méthode « ***parse »***, sert à convertir le Json en tableau JavaScript.

5 AJAX pour enregistrer les données

L'enregistrement part du principe suivant :

1. Parcourir ligne par ligne les données affichées (incluant la colonne cachée ID)
2. Stocker ces données dans un tableau à 2 dimensions (lignes/colonnes)
3. Coder ce tableau en JSON
4. Intégrer le JSON à une variable POST (la taille de la zone GET est limitée, POST ne l'est pas)
5. Envoyer la requête AJAX.

On détaille ici seulement le code des points 3-5.
La variable $res contient le tableau des données décrit ci-dessus :

```
              // Création d'un formulaire JS contenant les données
              var form = new FormData();
              form.append('res', JSON.stringify(res)); // Encodage Json pour le php

              // Envoi par Ajax à la page PHP (méthode POST)
              var xhr = new XMLHttpRequest();
              xhr.open('POST', 'phpajax.php?action=enregistrer', true);
              xhr.onload =  () => {
                     console.log(xhr.responseText);// une info sur le retour php
                     // Recharger après enregistrement
                     this.rechercher();
              };
              xhr.send(form); // Upload !
```

On remarque la méthode « ***stringify*** » de l'objet JSON, qui fait le contraire de la méthode « parse ».
On utilise un objet «FormData » du JavaScript pour obtenir un environnement de stockage identique à un formulaire HTML en méthode POST. Il suffit d'ajouter des couples nom/valeur qui seront récupérés dans le tableau $_POST[ ] côté PHP.
Ici nous n'avons besoin que d'une seule variable puisque toutes les données sont concentrées dans une structure JSON. C'est ce que fait la ligne `form.append('res', JSON.stringify(res));`

6 Le reste, c'est tout pareil…

Comme le reste n'est que réglage, tests de cohérences, répétitions, … je vous laisse les fichiers complets.

Fichier **index.html** :

```
<!DOCTYPE html>

<html>
	<head>
		<meta charset="utf-8"/>
		<meta name="viewport" content="width=device-width, initial-scale=1.0">
		<link rel="stylesheet" href="site.css" />
		<title>CMAV</title>
	</head>

	<body>
		<div class="enColonne contener">
			<div class="enLigne menuHaut">
					<input id="zRecherche" type="text" />
					<button id="btnRechercher">Rechercher</button>
			</div>
			<div class="enLigne affichage">
					<div id='zsql' class="enColonne sql"></div>
					<div class="enColonne menuDroite">
						<button id="btnAjouter">Ajouter</button>
						<button id="btnModifier">Modifier</button>
						<button id="btnEnregistrer">Enregistrer</button>
					</div>
			</div>
		</div>
		<script type="text/javascript" src="cl_cmav.js" > </script>
		<script type="text/javascript" src="principal.js" > </script>
	</body>
</html>
```

Fichier **site.css** :

```
div {
   display:flex;
   align-items:center;
   justify-content:space-around;
}

.enLigne { flex-direction:row; }
.enColonne { flex-direction:column; }

.contener {
   position:absolute; /*Il faut au moins un repère... */
   top:10%;
   left:0;/* puisqu'on n'utilise qu'une partie de la page */
   height:90%;
   width:100%;
   background-color:#5d5452;
}

.menuHaut {
   height:9%;
   width:98%;
   justify-content:center;
   background-color:lightGrey;
}

.affichage {
   height:89%;
   width:98%;
   background-color:lightGrey;
   /* flex-wrap: wrap-reverse; */
}

.menuDroite {
   height:98%;
   width:19%;
   background-color:blue; /*#00bcf9;*/
}

.sql {
   justify-content:center;
   height:98%;
   width:79%;
   background-color:#00bcf9;
}

.menuDroite button {
   width:50%;
   border-radius:10px;
}

.menuHaut button {
   width: 150px;
}

@media only screen and (max-width: 500px) {
   .contener { justify-content:flex-start; top:0; position:static; width:100%;}
   .menuHaut { height:60px; width:100%;}
   .affichage { height:auto; justify-content:flex-start; flex-direction:column; width:100%;}
   .sql { width:100%; min-height:400px; order:1; justify-content:flex-start;}
   .menuDroite { flex-direction:row; width:100%; height:60px;}
   .menuDroite button {width:25%;}
   .sql > div { width:100%; }
   .sql > div > input { width:30%; }
}
```