

EMF – Fribourg / Freiburg

Ecole des Métiers / Berufsfachschule

Technique / Technik

Module du 15.05.2023

au 12.11.2021

307 - Réaliser des pages Web interactives

Rapport personnel

Jeremy Monney

Date de création : 15.05.2023

Version 1 du 15.06.2023

Table des matières

1	INTRODUCTION.....	6
2	EXERCICE 1.....	6
2.1	DESCRIPTION.....	6
2.2	RÉALISATION.....	7
2.2.1	Onload	7
2.2.2	Onclick	7
2.2.3	Écouteur « click »	8
2.2.4	DOM	8
2.2.5	Chargement des scripts	8
2.2.6	CSS	9
3	EXERCICE 2.....	9
3.1	DESCRIPTION.....	9
3.1.1	Questions.....	10
3.2	RÉALISATION.....	11
3.2.1	Formulaire	11
3.2.2	validerUtilisateur()	11
3.2.3	Type des variables	12
3.2.4	Console chrome	12
3.2.5	CSS	14
4	EXERCICE 3.....	15
4.1	DESCRIPTION.....	15
4.2	RÉALISATION.....	15
4.2.1	Types de boutons programmables dans un formulaire HTML	17
5	EXERCICE 4.....	18
5.1	DESCRIPTION.....	18
5.2	RÉALISATION.....	19
6	EXERCICE 5.....	19
6.1	DESCRIPTION.....	19
6.2	RÉALISATION.....	20
6.2.1	Variables	20
7	EXERCICE 6.....	21
7.1	DESCRIPTION.....	21
7.2	RÉALISATION.....	22
8	EXERCICE 7.....	23
8.1	DESCRIPTION.....	23
8.2	RÉALISATION.....	24
8.2.1	For	24
8.2.2	While	25
8.2.3	DoWhile	25
9	EXERCICE 8.....	26
9.1	DESCRIPTION.....	26
9.2	RÉALISATION.....	27

10	EXERCICE 9	27
10.1	DESCRIPTION.....	27
10.2	RÉALISATION.....	28
11	EXERCICE 10 (PROGRAMMATION ORIENTÉE « OBJETS »).....	28
11.1	DESCRIPTION.....	28
11.2	RÉALISATION.....	29
12	EXERCICE 11 (PROGRAMMATION ORIENTÉE "CLASSE" EN JAVASCRIPT).....	33
12.1	DESCRIPTION.....	33
12.2	RÉALISATION.....	33
13	EXERCICE 12 (FONCTIONS ET IIFE EN JAVASCRIPT)	37
13.1	DESCRIPTION.....	37
13.2	RÉALISATION.....	37
13.2.1	IIFE	38
14	EXERCICE 13 (LES COOKIES).....	38
14.1	DESCRIPTION.....	38
14.2	RÉALISATION.....	39
15	EXERCICE 14 (BASES DE JQUERY)	41
15.1	DESCRIPTION.....	41
15.2	RÉALISATION.....	41
15.2.1	Tâches principales de jQuery	41
15.2.2	Sélecteurs	41
15.2.3	Ready()	41
15.2.4	Héritage	42
15.2.5	Différentes opérations	42
15.2.6	Lire fichier AJAX, Json	43
15.2.7	Charger un fichier HTML	43
15.2.8	Intégrer jQuery	43
16	EXERCICE 15 (PREMIÈRE UTILISATION DE JQUERY)	44
16.1	DESCRIPTION.....	44
16.2	RÉALISATION.....	45
17	EXERCICE 16	47
17.1	DESCRIPTION.....	47
17.2	RÉALISATION.....	47
18	EXERCICE 18	51
18.1	DESCRIPTION.....	51
18.2	RÉALISATION.....	51
19	EXERCICE 20	54
19.1	DESCRIPTION.....	54
19.2	RÉALISATION.....	56
20	REST	58
21	SOAP	58
22	GET	58
23	POST.....	58

24	PUT	58
25	DELETE	58
26	POSTMAN.....	59
27	PROJET	60
27.1	ANALYSE.....	60
27.1.1	Diagramme	60
27.1.2	Maquette	61
27.2	CONCEPTION	63
27.2.1	Diagramme de navigation.....	63
27.3	IMPLÉMENTATION.....	63
27.3.1	HTML	63
27.3.2	CSS	65
27.3.3	JS.....	65
27.3.3.1	httpService et viewService	65
27.3.4	Controller	68
27.4	TESTS.....	74
27.5	HÉBERGEMENT ET FONCTIONNEMENT	74
28	CONCLUSION.....	74

1 Introduction

Dans ce module nous allons apprendre le JavaScript et répéter HTML et CSS. Nous allons aussi faire la réalisation d'une application web à l'aide de JavaScript.

Voici les objectifs :

- Développer le caractère fonctionnel des pages Web interactives conformément aux données du problème.
- Développer une maquette pour la saisie et la présentation des données compte tenu des aspects ergonomiques.
- Choisir les éléments de formulaire appropriés pour la réalisation des données du problème, et garantir la validation des données entrées.
- Programmer l'application de manière modulaire et conformément aux directives de codification.
- Définir et en mettre œuvre des cas de tests appropriés pour des pages Web interactives et documenter dans le procès-verbal de tests.

2 Exercice 1

2.1 Description

Pour ce premier exercice, veuillez créer ou copier, avec votre outil de développement, les sources contenues dans le fichier « exercice_1.zip » ci-dessous dans le dossier principal « /exercices/exercice_1 ».

Adaptez le texte (votre nom et prénom) et sa mise en forme (gras) ainsi que la couleur du bouton 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

Cliquez sur le bouton qui contient votre prénom!

Teste-moi, James

C'est **James Bond** qui a pressé le bouton !

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.

Hébergement

Adaptez les en-têtes de vos différents fichiers et uploader votre exercice 1 sur votre domaine personnel de l'école.

2.2 Réalisation

2.2.1 Onload

D'abord on utilise le Onload dans le body. Cet élément va exécuter une fonction au chargement de la page. Normalement on utilise le Onload sur le body. Voici le Onload dans l'exercice qui exécute la fonction initCtrl :

```
<body onload="initCtrl()">
  <div id="container">
    <p>Cliquez sur le bouton qui contient votre prénom (js) !</p>
    <button id="testez">Teste-moi, Jeremy</button>
    <p id="info">&nbsp;</p>
  </div>
</body>
```

2.2.2 Onclick

Le Onclick permet d'exécuter une fonction quand on clique sur un élément.

Voici le bouton dans html de l'exercice qui utilise Onclick et qui va exécuter la fonction testez2:

```
<body>
  <div id="container">
    <p>Cliquez sur le bouton qui contient votre prénom (html) !</p>
    <button id="testez" onclick="testez2()">Teste-moi, Jeremy</button>
    <p id="infos">&nbsp;</p>
  </div>
</body>
```

Et voici la fonction JS qui va afficher le texte en dessous du bouton :

```
function testez2() {
  document.getElementById("infos").innerHTML =
    "C'est <b>Jeremy Monney</b> qui a pressé le bouton !";
}
```

2.2.3 Écouteur « click »

L'écouteur click ne va rien faire et juste « écouter » jusqu'à ce qu'un bouton (par rapport un ID) va être cliquer et ensuite exécuter une fonction.

Dans notre cas l'écouteur va être exécuter pendant le chargement de la page (Onload). En suite il exécute la fonction testez quand on clique un bouton avec l'ID « testez ».

Voici la fonction avec l'écouteur :

```
function initCtrl() {
  // Ecouteur du bouton "Testez-moi..."
  document.getElementById("testez").addEventListener("click", testez);
}
```

2.2.4 DOM

Le DOM (Document Object Model) permet d'accéder et manipuler les éléments HTML, les attributs, le texte et le contenu d'une page web avec JS.

On peut accéder aux éléments HTML du DOM en utilisant différentes méthodes et propriétés. Par exemple, on peut utiliser `document.getElementById('id')` pour obtenir un élément par son ID.

Le DOM te permet également de gérer les événements (clics, survols, soumissions de formulaires, etc.) en utilisant des écouteurs d'événements comme `element.addEventListener('click', maFonction)`.

2.2.5 Chargement des scripts

Les scripts JavaScript peuvent être chargés dans différentes parties d'une page web, en fonction des besoins et des objectifs spécifiques. Voici les endroits courants où les scripts JavaScript sont généralement chargés :

- Dans la balise `<head>` : Les scripts peuvent être placés dans la balise `<head>` du document HTML en utilisant la balise `<script>`. Cela permet de charger le script avant le contenu de la page, ce qui peut être utile si le script doit être exécuté dès le chargement de la page.

```
<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>
```

- En tant que script inline : Les scripts JavaScript peuvent également être inclus directement dans le code HTML en utilisant l'attribut `onload`, `onclick`, `onsubmit`, etc. Cela est souvent utilisé pour des scripts de petite taille qui sont spécifiques à un élément ou à une interaction particulière.

2.2.6 CSS

Voici le CSS de l'exercice (testez est le bouton):

```
body {
  background-color: grey;
}

#container {
  background-color: white;
  width: 90%;
  padding: 1em;
  border: 1px solid black;
  margin: auto;
  font-family: Verdana, Arial, serif;
}

#testez{
  color: rgb(144, 0, 211);
}
```

3 Exercice 2

3.1 Description

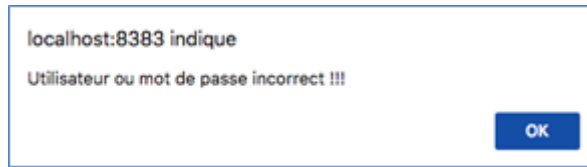
On vous demande de compléter le conteneur pour afficher un composant formulaire (form) HTML5 pour simuler une opération de login. Pour le moment, le test du login se fera localement en JavaScript (valides : « admin » quelle que soit la casse pour l'utilisateur et « emf123 » pour le mot de passe).

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.

Maquette :

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) :



La fonction Javascript alert() ouvre un popup d'information, confirm() ouvre un popup de confirmation ou non et prompt() ouvre un popup d'input, qui permet à l'utilisateur de saisir du texte.

3.1.1 Questions

À quoi sert l'attribut « placeholder » ?

L'attribut placeholder est utilisé dans les éléments de formulaire HTML (comme les zones de texte et les champs d'entrée) pour afficher un texte indicatif ou un exemple de ce qui est attendu dans le champ, avant que l'utilisateur ne saisisse une valeur.

À quoi sert l'attribut « autofocus » ?

L'attribut "autofocus" est utilisé pour spécifier qu'un champ de saisie doit être automatiquement mis en surbrillance et avoir le focus lorsque la page est chargée.

Quelle est la différence entre le bouton de l'exercice 1 et celui-ci ?

Dans l'exercice on utilise l'élément « button » mais dans cet exercice on crée un input du type « button ». Le bouton input est utilisé pour les formulaires et il est généralement utilisé lorsque le bouton doit simplement déclencher une action sans avoir de contenu textuel supplémentaire. Les deux boutons exécutent une fonction Onclick.

3.2 Réalisation

3.2.1 Formulaire

Avec les éléments input on peut créer des « textFields » on leur donne un ID pour après pouvoir récupérer les données. On peut après récupérer les données des inputs à l'aide de JS avec la manière suivante :

```
let username = document.getElementById("username").value;
let pwd = document.getElementById("password").value;
```

Avec le .value de document.getElementById on peut récupérer la valeur des éléments HTML. On les stocke dans une variable avec le mot de clé « let » et le nom de la variable.

Voici le formulaire HTML :

```
<body>
  <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="password">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>
</body>
```

3.2.2 validerUtilisateur()

Cette fonction va tester si l'utilisateur et le mdp passe dans l'input et correcte. Dans un if il va tester les variables. Avec l'aide de « window » on peut afficher des pop-up messages. Il y a le message alert qui va envoyer un pop-up avec un message et un bouton ok :

127.0.0.1:5500 indique
Utilisateur ou mot de passe incorrect !!!



Et il y a aussi le message confirm qui affiche un message et on peut cliquer sur le bouton ok ou annuler :

127.0.0.1:5500 indique
Validation OK



Avec `console.log` on peut afficher des messages dans la console chrome (Voici point 3.2.4 Console chrome).

Voici la fonction :

```
function validerUtilisateur() {  
  //Creation des variables  
  let username = document.getElementById("username").value;  
  let pwd = document.getElementById("password").value;  
  
  //teste login  
  if (username == "admin" && pwd == "Pa$$w0rd") {  
    window.confirm("Validation OK");  
    console.log("Validation OK");  
  } else {  
    window.alert("Utilisateur ou mot de passe incorrect !!!");  
    console.log("Utilisateur ou mot de passe incorrect !!!");  
  }  
}
```

3.2.3 Type des variables

Dans JS on n'a pas besoin de définir le type de la variable. JS va automatiquement lui donner un type. Mais JS comprends ces différents types :

- Nombre (Number) : Représente les valeurs numériques, qu'elles soient entières ou à virgule flottante.
- Chaîne de caractères (String) : Représente une séquence de caractères, tels que du texte, entourée de guillemets simples (") ou doubles (").
- Booléen (Boolean) : Représente une valeur de vérité, soit vrai (true) ou faux (false). Il est souvent utilisé dans les conditions et les opérations de contrôle.
- Null : Représente l'absence intentionnelle de toute valeur. Elle est utilisée pour indiquer l'absence d'une valeur valide.
- Indéfini (Undefined) : Représente une variable qui n'a pas été définie ou à laquelle aucune valeur n'a été attribuée.
- Objet (Object) : Représente une collection de données et de fonctionnalités associées. Les objets sont des structures de données complexes, pouvant être créés à partir d'un modèle (prototype) ou être des instances de classes.
- Tableau (Array) : Représente une collection ordonnée d'éléments, qui peuvent être de différents types de données. Les tableaux sont indexés par des entiers et offrent des méthodes pour manipuler et accéder à leurs éléments.
- Fonction (Function) : Représente un bloc de code réutilisable, pouvant prendre des paramètres en entrée, effectuer des calculs et renvoyer une valeur.
- Symbole (Symbol) : Représente une valeur unique et immuable, souvent utilisée comme identifiant pour les propriétés des objets.

3.2.4 Console chrome

Pour ouvrir la Console dans Chrome on peut faire une clique droite, en suite il faut aller sur inspecter et en fin en haut on peut cliquer sur l'onglet Console. Et comme ça on peut afficher la console dans chrome qui va permettre d'afficher des erreurs etc.

3.2.5 CSS

Tous qui est user-form est le style du formulaire. Field est pour tous le formulaire, label pour le texte a cote des inputs et inputs pour les texte fields.

Voici le CSS de l'exercice :

```
#container {
  background-color: white;
  width: 90%;
  padding: 1em;
  border: 1px solid black;
  margin: auto;
  font-family: Verdana, Arial, serif;
}

.user-form .field{
  padding: 10px;
  background-color: lightgrey;
}

.user-form .field label{
  display: inline-block;
  width: 130px;
}

.user-form .field input{
  display: inline-block;
}

#valider{
  border-radius: 10px;
  border-style: none;
  background-color: lawngreen;
  margin-top: 10px;
  width: 100px;
}
```

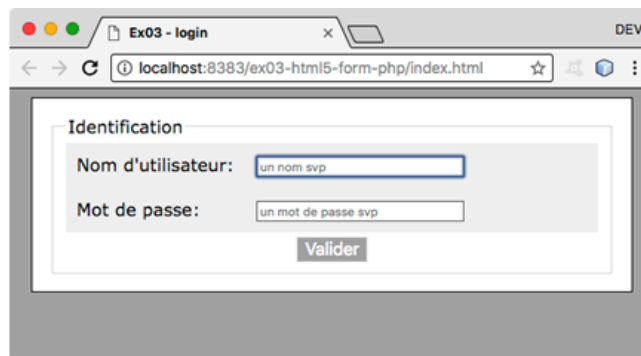
4 Exercice 3

4.1 Description

Le but de l'exercice est :

- 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://galley.pmf-informatique.ch>.

Maquette



4.2 Réalisation

Dans l'exercice on utilise une fois le POST et une fois le GET. Ils permettent de récupérer les informations de l'input en deux différentes manières. Pour le stocker dans une variable on doit juste déclarer la variable et écrire `$_POST/GET['name']` (le name de l'élément input). Pour mettre tout en minuscule on peut encore utiliser la fonction « `strtolower` » qui va transformer tout en minuscule. Le `echo` va ensuite permettre d'afficher les informations reçues et il est concaténé avec les variables et du HTML (`
` qui permet de faire un retour à la ligne). On utilise un « `.` » pour concaténer.

GET est principalement utilisé pour récupérer des données et est visible dans l'URL, tandis que POST est utilisé pour envoyer des données sensibles ou volumineuses et n'est pas visible dans l'URL (mais dans l'inspection Chrome). Il est important de choisir la méthode appropriée en fonction des besoins spécifiques de l'application pour assurer la sécurité des données échangées entre le client et le serveur.

Dans le test, si il y a encore un peu de JS dans les balises `script` pour faire le pop-up.

Voici le PHP avec POST :

```
<?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>";
    }
}
?>
```

Voici le PHP avec GET :

```
<?PHP
// test si on a reçu une donnée de formulaire nommée "username"
if (isset($_GET['username'])) {

    // récupération des données transmises dans des variables locales
    $username = strtolower($_GET['username']);
    $password = $_GET['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>";
    }
}
?>
```


Dans le formulaire dans HTML on doit encore définir quelle méthode on utilise pour récupérer les données (GET ou POST).

Voici le HTML :

```
<body>
  <form class="user-form" action="php/login.php" method="POST">
    <fieldset>
      <legend>Identification:</legend>
      <div class="field">
        <label for="username">Nom d'utilisateur:</label>
        <input type="text" size="30" name="username" id="username"
          placeholder="un nom svp" autofocus />
      </div>
      <div class="field">
        <label for="password">Mot de passe:</label>
        <input type="password" size="30" name="password" id="password"
          placeholder="un mot de passe svp" />
      </div>
      <div class="button">
        <input type="submit" value="Valider">
      </div>
    </fieldset>
  </form>
</body>
```

4.2.1 Types de boutons programmables dans un formulaire HTML

Il y a différents types de boutons dans HTML.

Voici encore ce que nous résumer pour les types de boutons les plus courants :

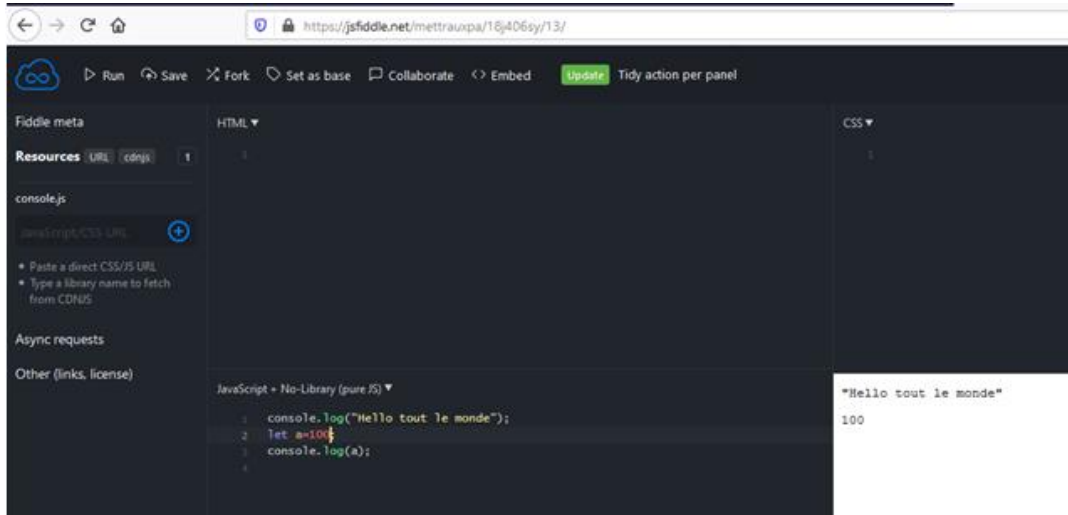
- Type="button" : ce type crée un simple bouton cliquable sans fonctionnalité particulière.
- Type="submit" : ce type crée un bouton qui envoie un formulaire vers un serveur lorsqu'il est cliqué. Le visiteur sera conduit à la page indiquée dans l'attribut action.
- Type="reset" : ce type crée un bouton qui réinitialise les valeurs d'un formulaire à leurs valeurs par défaut.
- Type="image" : ce type crée un bouton qui est représenté par une image spécifiée dans l'attribut src.
- Type="checkbox" : ce type crée un bouton qui peut être coché ou décoché.
- Type="radio" : ce type crée un bouton qui peut être sélectionné dans un groupe de boutons liés.
- Type="file" : ce type crée un bouton qui permet à l'utilisateur de sélectionner un fichier sur son ordinateur.

5 Exercice 4

5.1 Description

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 :

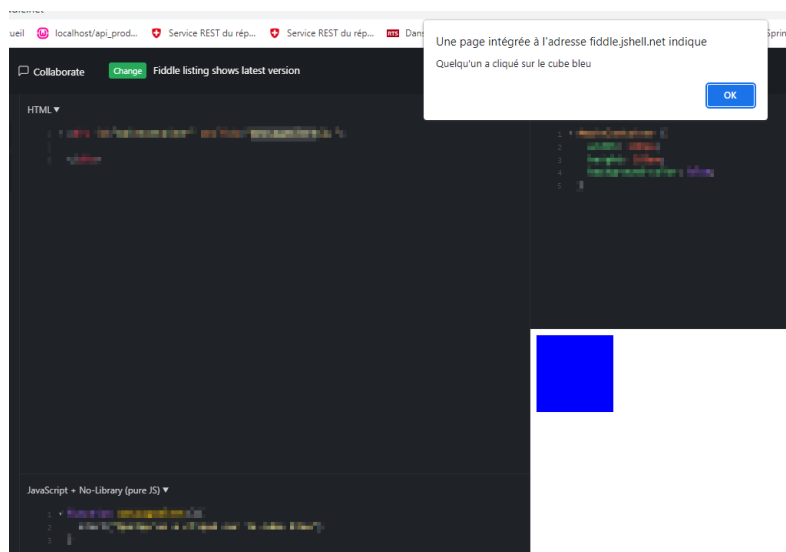
- De créer un compte gratuit chez eux (pour pouvoir sauver ou partager des fiddle)
- De tester un premier fiddle avec un affichage du fameux « hello world » dans un popup JavaScript (ce que vous savez déjà faire) :



Vous pouvez noter le codage très souple, puisqu'aucune balise HTML n'a précisé que nous utilisions du JavaScript.

L'avantage de cette solution est qu'il est possible de tester du code HTML, CSS ou Javascript rapidement sans devoir mettre en place des fichiers et ouvrir son IDE.

Créez ensuite une petite application Web. Grâce à de l'HTML et du CSS, créez un carré bleu de 100px de côté. Implémentez ensuite une fonction `messageAlert` qui affichera un popup avec le message "Quelqu'un a cliqué sur le carré bleu" lorsqu'on clique sur le carré contenu dans le rendu de la page.



5.2 Réalisation

JSFiddle est un outil en ligne gratuit qui permet aux développeurs d'écrire, d'exécuter et de partager du code en HTML, CSS et JavaScript de manière interactive.

Voici les avantages d'utiliser JSFiddle :

- Environnement de développement
- Partage facile
- Prise en charge des bibliothèques externes
- Possibilité de tester rapidement du code

Voici mon Fiddle pour cet exercice :

<https://jsfiddle.net/monneyj/8q3w0ep9/21/>

6 Exercice 5

6.1 Description

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

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 ``...${...}...``)
9. Compléter en faisant de même pour une soustraction, une multiplication et une division des deux variables ;
10. Stocker « Bonjour » dans la variable a ;
11. Stocker « les amis » dans la variable b ;
12. Afficher « bonjour les amis » dans la console en concaténant les variables ;
13. Faites la même chose en utilisant un littéral avec ``...${...}...`` ;
14. Stocker « true » dans la variable a ;
15. Stocker « false » dans la variable b ;
16. Effectuer une opération AND entre les 2 variables et afficher le résultat sous cette forme « true AND false = ? » ;
17. Effectuer une opération OR entre les 2 variables et afficher le résultat sous cette forme « true OR false = ? » ;
18. Stocker la date du jour dans la variable a avec `new Date()`;

19. Calculer une nouvelle date dans la variable b qui est 61 jours avant la date courante (utilisation getDate, setDate)
20. Afficher les dates contenues dans les variables a et b en vous aidant de « toLocaleString(), toLocaleDateString() et toLocaleTimeString() ». Afficher la date et l'heure, la date uniquement et l'heure uniquement.
21. Le mot réservé « typeof » permet de connaître le type utilisé momentanément pour une variable. Stocker la valeur de Math.PI dans a, « bonjour » dans b, créer et assigner true dans c, créer, assigner la date courante dans d et déclarez la variable e sans rien lui affectez, puis afficher le type pour les 5 variables.

6.2 Réalisation

Voici le lien du fiddle :

<https://jsfiddle.net/monneyj/rzbkua1c/17/>

6.2.1 Variables

En JavaScript, il existe trois types de variables : var, let et const.

- La déclaration avec « var » crée une variable à portée de fonction ou globale. Elle peut être réaffectée et redéclarée dans la même portée.
- La déclaration avec « let » crée une variable à portée de bloc (par exemple, une boucle ou un bloc if). Elle peut être réaffectée, mais pas redéclarée dans la même portée.
- La déclaration avec « const » crée une variable à portée de bloc, dont la valeur est constante et ne peut pas être réaffectée ni redéclarée dans la même portée.

Il est recommandé d'utiliser let et const plutôt que var pour des raisons de cohérence et de prévention des erreurs.

On peut aussi déclarer des variables sans mots de clés.

Pour savoir le type de la variable on peut utiliser la fonction typeof(). Voici un exemple :

```
a = Math.PI;
b = "bonjour";
let c = true;
let d = new Date();
let e;
//Affiche le type utilisé momentanément pour une variable
console.log(`${typeof(a)}, ${typeof(b)}, ${typeof(c)}, ${typeof(d)},
${typeof(e)}`);
```

On peut utiliser des template littéral pour incorporer des variables dans une chaîne de caractères en utilisant \${variable}. Voici un exemple :

```
console.log(`${a} + ${b} = ${a+b}`);
```

Résultat :

```
"15 + 9 = 24"
```

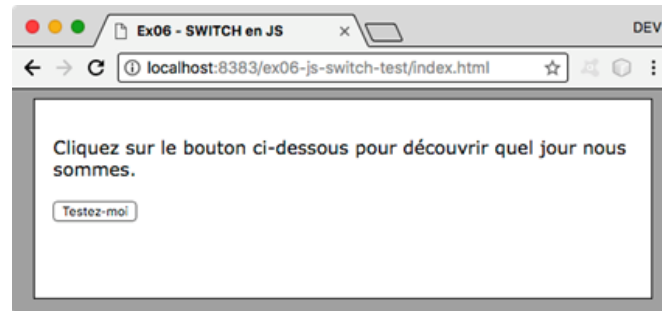
7 Exercice 6

7.1 Description

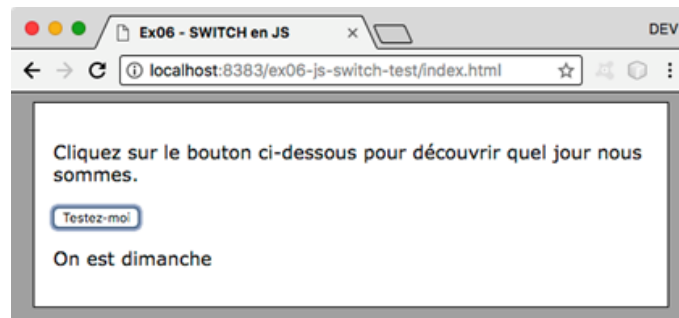
Nous avons déjà testé le « IF », testons maintenant le « SWITCH », avec une petite application qui doit afficher le jour de la semaine directement en remplacement du contenu d'une balise HTML existante.

Maquette

Au démarrage de l'application :

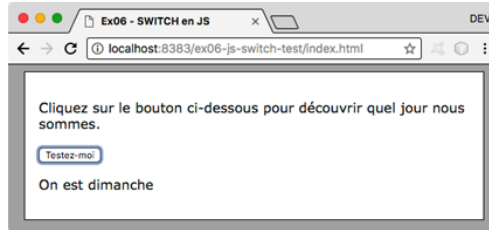


Après le clic, affichage du jour de la semaine :



7.2 Réalisation

Le « `new Date().getDay()` » retourne un nombre entre 0-6. 0 = dimanche, 1 = lundi etc. Du coup avec le Switch on peut après crée la variable `day` avec le nom du jour et a la fin on l'affiche dans un `<p>`. Cette méthode s'exécute quand on clique sur le bouton.



Voici le Switch :

```
function afficherJourSemaine(){
  switch (new Date().getDay()) {
    case 0:
      day = "Dimanche";
      break;
    case 1:
      day = "Lundi";
      break;
    case 2:
      day = "Mardi";
      break;
    case 3:
      day = "Mercredi";
      break;
    case 4:
      day = "Jeudi";
      break;
    case 5:
      day = "Vendredi";
      break;
    case 6:
      day = "Samedi";
  }
  document.getElementById("info").innerHTML = "On est " +day;
}
```

Ici on a fait la même chose mais avec un tableau. D'abord j'ai créé un tableau « `const days = [...]` » avec tous les jours. En suite dans une variable je stocke le jours nouveau avec « `new Date().getDay()` » je reçois un int et je peux stocke du coup le jours dans la variable `day` en récupérant le jours dans le tableau avec `days[position en int]`. A la fin j'affiche nouveau le jours dans un `<p>`.

Voici la version avec tableau :

```
function afficherJourSemaineTab(){
  const days = ["Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi",
    "Samedi"];
  day = days[new Date().getDay()];
  document.getElementById("infos").innerHTML = "On est " +day + " via
    (tableau)";
}
```

Si je veux afficher tous les jours dans le tableau je devrais faire un forEach:

```
days.forEach(day => console.log(day));
```

8 Exercice 7

8.1 Description

Au démarrage de l'application :

Cliquez sur les boutons ci-dessous pour tester les différentes sortes de boucles en JavaScript.

Cliquez sur les boutons ci-dessous pour tester les différentes sortes de boucles en JavaScript.

```
for (let i=0; i<5; i++){...}
i = 0
i = 1
i = 2
i = 3
i = 4
--> À utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)
```

Cliquez sur les boutons ci-dessous pour tester les différentes sortes de boucles en JavaScript.

```
while (i<5) {...}
i = 0
i = 1
i = 2
i = 3
i = 4
--> À utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle
```

Cliquez sur les boutons ci-dessous pour tester les différentes sortes de boucles en JavaScript.

```
do {...} while (i<5);
i = 0
i = 1
i = 2
i = 3
i = 4
--> À utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle mais avec un passage obligatoire
```

8.2 Réalisation

D'abord j'ai créé une méthode avec des écouteurs pour les 3 boutons :

```
function initCtrl() {
  // Ecouteur du bouton "for, while et do"
  document.getElementById("for").addEventListener("click", testerFor);
  document.getElementById("while").addEventListener("click", testerWhile);
  document.getElementById("do").addEventListener("click", testerDoWhile);
}
```

Et voici le HTML qui exécute la méthode au chargement :

```
<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>
```

8.2.1 For

On utilise un for quand on sait que l'on veut itérer x fois (x connu avant de commencer la boucle). Dans les paranètes on doit d'abord déclarer la variable, mettre une condition et incrémenter i. En suite pour chaque itération on affiche la valeur de i. Jusque a ce que i vaut 4.

Voici la fonction For :

```
function testerFor(){
  document.getElementById("info").innerHTML = "for (let i = 0; i < 5; i++){...}</br>"
  for (let i = 0; i < 5; i++) {
    document.getElementById("info").innerHTML += "i = " + i + "</br>";
  }
  document.getElementById("info").innerHTML += "---> A utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)";
}
```


8.2.2 While

On utilise le While quand on ne sait pas le nombre d'itération au démarrage de la boucle. D'abord on doit déclarer une variable avec la valeur 0. En suite dans le While on doit dire la condition. Après on affiche la valeur de i et on incrémente la valeur de i. Et on répète ça jusqu'à ce que i vaut 4.

Voici la fonction While :

```
function testerWhile(){
  document.getElementById("info").innerHTML = "while (i<5){...}</br>"
  let i = 0;
  while(i<5){
    document.getElementById("info").innerHTML += "i = " + i + "</br>";
    i++;
  }
  document.getElementById("info").innerHTML += "----> A utiliser si on ne sait pas le nombre de iteration au demarrage de la boucle";
}
```

8.2.3 DoWhile

On utilise DoWhile quand on ne sait pas le nombre d'itération au démarrage de la boucle mais avec un passage obligatoire. Comme dans la boucle while on doit d'abord déclarer une variable avec la valeur 0. Ensuite il va afficher au moins une fois la valeur de i et après il teste la condition jusqu'à ce que i est 4.

Voici la fonction DoWhile :

```
function testerDoWhile(){
  document.getElementById("info").innerHTML = "do{...} while(i<5)</br>"
  let i = 0;
  do {
    document.getElementById("info").innerHTML += "i = " + i + "</br>";
    i++;
  } while (i<5);
  document.getElementById("info").innerHTML += "----> A utiliser si on ne sait pas le nombre de iteration au demarrage de la boucle mais avec un passage obligatoire";
}
```

9 Exercice 8

9.1 Description

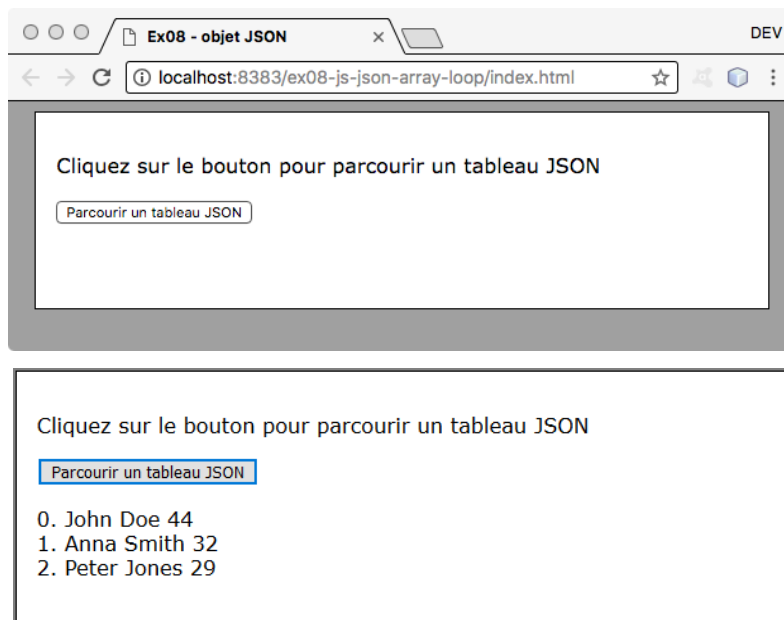
Nous allons tester une imbrication de 2 boucles sur les données d'un tableau de personnes défini par un objet de type JSON (JavaScript Object Notation).

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.

Maquette



9.2 Réalisation

D'abord on crée un tableau Json de personnes, ça veut dire que le tableau contient des personnes et chaque personne a un prénom, nom et un âge. En suite on fait un for et pour chaque personne on affiche son prénom, nom et son âge et on affiche ça dans un <p>.

Cette fonction se lance Onclick.

Voici la fonction :

```
function parcourirUnTableauJSON(){
  const personnes = [
    {prenom: "John", nom: "Doe", age: 44},
    {prenom: "Anna", nom: "Smith", age: 32},
    {prenom: "Peter", nom: "Jones", age: 29}
  ] ;

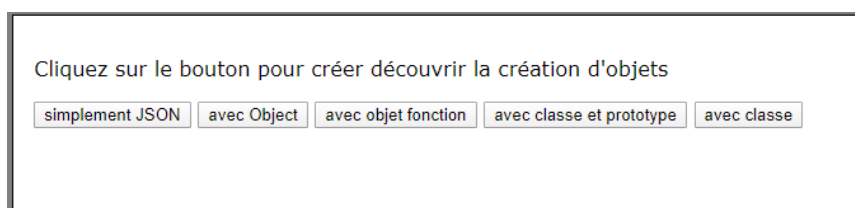
  let res = "";
  for (let i = 0; i < personnes.length; i++) {
    res += i + ". " + personnes[i].prenom + " " + personnes[i].nom + " "
    + personnes[i].age + "<br>";
  }

  document.getElementById("info").innerHTML = res;
}
```

10 Exercice 9

10.1 Description

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.



10.2 Réalisation

La création d'un objet avec une classe est comme en Java. La classe est comme le plan de construction pour l'objet.

Voici comment on crée une classe :

```
class Eleve {
  constructor(prenom, nom, age) {
    this.prenom = prenom;
    this.nom = nom;
    this.age = age;
  }

  toString() {
    return this.prenom + " " + this.nom + " (" + this.age + ")";
  }
}
```

En suite on peut créer l'objet dans une fonction. On crée l'objet avec le mot de clé « new » et ensuite on doit définir les paramètres de l'objet. Voici comment crée des objets et les affiche avec la méthode toString:

```
function creerDesObjetsClasse() {
  console.log("-----Class-----");
  let p1 = new Eleve("Julien", "Tartampion", 18);
  console.log(p1);
  let p2 = new Eleve("Julia", "Tartampion", 22);
  console.log(p2);
  let txt = p1 + "<br>" + p2;
  document.getElementById("info").innerHTML = txt;
}
```

11 Exercice 10 (Programmation orientée « objets »)

11.1 Description

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

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

Veuillez introduire une nouvelle personne ou sélectionner une existante :

Prénom:

Nom:

Âge:

- [Doe John \(44\)](#)
- [Jones Peter \(29\)](#)
- [Smith Anna \(32\)](#)

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

- [Aebischer Jean \(33\)](#)
- [Doe John \(44\)](#)
- [Jones Peter \(29\)](#)
- [Smith Anna \(32\)](#)
- [Zbinden Michel \(45\)](#)

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.

11.2 Réalisation

Ce projet contient 3 fichiers JS. Un indexCtrl qui gère la vue et le contrôleur, le bean personnes qui contient la classe de la personne et le worker qui fait tous la logique.

Voici le HTML qui :

```
<body>
  <div id="container">
    <form class="user-form">
      <fieldset>
        <legend>Veuillez introduire une nouvelle personne ou sélectionner
une existante    </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">
      </fieldset>
    </form>
  </div>
  <ul>
    <li>• <a href="#">Doe John (44)</a></li>
    <li>• <a href="#">Jones Peter (29)</a></li>
    <li>• <a href="#">Smith Anna (32)</a></li>
  </ul>
```

```

    </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>

```

Le bean `personne` est assez simple, elle contient un constructeur avec trois attributs, et une méthode « `toString()` ».

Voici le bean `personne` :

```

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

Personne.prototype.toString = function() {
  return `${this.prenom} ${this.nom} (${this.age})`;
}

```

La fonction privée « `_trouverPersonne` » vas comparer le `toString` des personnes et retourne l'index si c'est la même personne sinon elle retourne -1. La méthode `ajouterPersonne` utilise la fonction « `push` » pour ajouter une personne dans le tableau. La fonction « `splice` » permet de supprimer un ou plusieurs éléments d'une liste depuis un index donné.

Voici le worker :

```

// 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; i<personnes.length;i++){

```

```

        if(personnes[i].toString() == p.toString()){
            idx = i;
        }
    }
    return idx;
}

// ajouter une personne dans la liste des personnes si pas trouvée
function ajouterPersonne(p) {
    let idx = _trouverPersonne(p);
    if(idx == -1){
        personnes.push(p);
        personnes.sort();
    }
}

// supprimer une personne dans la liste des personnes si trouvée
function supprimerPersonne(p) {
    let idx = _trouverPersonne(p);
    if(idx > -1){
        personnes.splice(idx, 1);
    }
}

```

Le indexCtrl est repartie en 3 parties.

D'abord on utilise le « onreadystatechange » qui vas appeler la fonction si tous le DOM est charge. Et il va afficher toutes les personnes.

Dans la deuxième partie on va afficher et écrire toutes les personnes. On les écrit dans une liste . En suite on les affiche dans les inputs. Et la dernière fonction lit toutes les informations dans les inputs pour crée l'objet personne.

La dernière partie est là pour utilise les fonctions pour supprimer/ajouter et sélectionner une personne.

Voici le indexCtrl :

```

/*
 * 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>";
    for (let i = 0; i < personnes.length; i++) {
        txt += "<li><a class='link' href='#' onclick='selectionnerPersonne(" + i
+ ")'>" + personnes[i].toString() + "</a> </li>";
    }
    txt += "</ul>";
    document.getElementById("info").innerHTML = txt;
}

// affiche les infos d'une personne dans le formulaire
function _afficherInfosPersonne(p) {
    document.getElementById("prenom").value = p.prenom;
    document.getElementById("nom").value = p.nom;
    document.getElementById("age").value = p.age;
}

// lit le contenu des masques de saisie pour en faire une personne
function _lireInfosPersonne() {
    let p = null;
    let prenom = document.getElementById("prenom").value;
    let nom = document.getElementById("nom").value;
    let age = document.getElementById("age").value;
    if (prenom && nom && age) {
        p = new Personne(prenom, nom, age);
    }
    return p;
}

/*
 * 3. METHODES PUBLIQUES NECESSAIRES A LA VUE
 */

// appelée depuis la vue pour afficher les données de la personne
sélectionné
function selectionnerPersonne(i) {
    _afficherInfosPersonne(personnes[i]);
}

// appelée depuis la vue pour ajouter une personne
function ajouter() {
    let p = _lireInfosPersonne();
    ajouterPersonne(p);
    _afficherPersonnes();
}

```



```
// appelée depuis la vue pour supprimer une personne
function supprimer() {
  let p = _lireInfosPersonne();
  supprimerPersonne(p)
  _afficherPersonnes()
}
```

12 Exercice 11 (Programmation orientée "classe" en Javascript)

12.1 Description

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 !

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.

12.2 Réalisation

Dans le HTML on a juste dû changer la notation de la fonction qu'on lance avec le onClick :

```
<div class="button">
  <input type="button" value="Ajouter" onclick="ctrl.ajouter();">
  <input type="button" value="Supprimer" onclick="ctrl.supprimer();">
  <input type="reset" value="Nettoyer">
</div>
```

Voici le bean Personne :

```
class Personne {
  constructor(prenom, nom, age) {
    this.prenom = prenom;
    this.nom = nom;
    this.age = age;
  }

  toString() {
    return `${this.prenom} ${this.nom} (${this.age})`;
  }
}
```

```

}
}

```

Dans le Worker on n'a pas du change grand-chose. Il faut juste utilise le « this » pour utilise les fonctions. Ca veut dire au lieu de personnes.sort() on doit faire this.personnes.sort(). Quand il y a un teste avec trois = le teste comparer pas que la valeur mais aussi le type.

Voici le Worker :

```

// définition du modèle de données, créé au chargement du js dans le
index.html
class Worker {
  constructor() {
    this.personnes = [
      new Personne("John", "Doe", 44),
      new Personne("Anna", "Smith", 32),
      new Personne("Peter", "Jones", 29),
    ];

    // premier tri de la liste de personnes
    this.personnes.sort();
  }

  // fonction privée pour retrouver l'index d'une personne dans le tableau, -1
  autrement
  // il faut comparer avec toString()
  trouverPersonne(p) {
    let idx = -1;
    for (let i = 0; i < this.personnes.length; i++) {
      if (this.personnes[i].toString() == p.toString()) {
        idx = i;
      }
    }
    return idx;
  }

  // ajouter une personne dans la liste des personnes si pas trouvée
  ajouterPersonne(p) {
    let idx = this.trouverPersonne(p);
    if (idx == -1) {
      this.personnes.push(p);
      this.personnes.sort();
    }
  }

  // supprimer une personne dans la liste des personnes si trouvée
  supprimerPersonne(p) {
    let idx = this.trouverPersonne(p);
    if (idx > -1) {
      this.personnes.splice(idx, 1);
    }
  }
}

```

```

    }
  }
}

```

Avec le IndexCtrl la même chose comme avec le Worke, on utilise this.wrk...

Voici le IndexCtrl :

```

/*
 * 1. DOM PRET : DEMARRAGE DE L'APPLICATION
 */
class Ctrl {
  constructor() {
    this.wrk = new Worker();
  }

  // affiche la liste des données au bas de la vue (avec du HTML généré)
  afficherPersonnes() {
    let txt = "<ul>";
    for (let i = 0; i < this.wrk.personnes.length; i++) {
      let p = this.wrk.personnes[i];
      txt += `<li><a href="#"
onclick="ctrl.selectionnerPersonne(${i})">${p.toString()}</a></li>`;
    }
    txt += "</ul>";
    document.getElementById("info").innerHTML = txt;
  }

  // affiche les infos d'une personne dans le formulaire
  afficherInfosPersonne(p) {
    document.getElementById("prenom").value = p.prenom;
    document.getElementById("nom").value = p.nom;
    document.getElementById("age").value = p.age;
  }

  /*
   * 2. METHODES PRIVEES DE LECTURE/ECRITURE DANS LA VUE
   */
  // lit le contenu des masques de saisie pour en faire une personne
  lireInfosPersonne() {
    let p = null;
    let prenom = document.getElementById("prenom").value;
    let nom = document.getElementById("nom").value;
    let age = document.getElementById("age").value;
    if (age && nom && prenom) {
      p = new Personne(prenom, nom, age);
    }
    return p;
  }
}

```

```

/*
 * 3. METHODES PUBLIQUES NECESSAIRES A LA VUE
 */
// appelée depuis la vue pour afficher les données de la personne
sélectionnné
selectionnerPersonne(i) {
    this.afficherInfosPersonne(this.wrk.personnes[i]);
}

// appelée depuis la vue pour ajouter une personne
ajouter() {
    let p = this.lireInfosPersonne();
    this.wrk.ajouterPersonne(p);
    this.afficherPersonnes();
}

// appelée depuis la vue pour supprimer une personne

supprimer() {
    let p = this.lireInfosPersonne();
    this.wrk.supprimerPersonne(p);
    this.afficherPersonnes();
}
}

document.onreadystatechange = function () {
    if (document.readyState === "complete") {
        window.ctrl = new Ctrl();
        window.ctrl.afficherPersonnes();
    }
};

```

13 Exercice 12 (Fonctions et IIFE en JavaScript)

13.1 Description

Dans cet exercice on a du tester les différentes manières d'écrire des fonctions dans JS.

13.2 Réalisation

Une fonction est premièrement définie et ensuite exécutée une à plusieurs fois. Voici trois manières d'écrire une fonction JS (Voici les commentaires):

```
//Déclaration d'une fonction
function a(add) {
    let val = 1;
    console.log(val + add) ;
}
a(3);

// Déclaration d'une expression fonction (fonction anonyme)
let b = function(add) {
    let val = 2;
    console.log(val + add) ;
} ;
b(2);

// Déclaration d'une fonction flèche (arrow function)
(function(add) {
    let val = 3;
    console.log(val + add) ;
})(1) ;

((add) => {
    let val = 4;
    console.log(val + add) ;
})(2) ;
```

13.2.1 IIFE

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
```

14 Exercice 13 (Les cookies)

14.1 Description

Par l'intermédiaire de cookies, il est possible d'écrire sur le poste de l'internaute des informations de manière permanente. Les cookies qui se présentent comme de petits fichiers texte peuvent entreposer un nombre limité de données mais cette technique peut se révéler précieuse pour conserver des contenus de variables mémoires réutilisables au fil de la navigation au travers des nombreuses pages de votre site web (conservation d'identifiants, de mots de passe même si cela est dangereux, de préférences utilisateur...).

Il est à noter qu'il est possible d'interdire dans la grande majorité des navigateurs que les cookies soient utilisables. Il est vrai que cette technique est relativement intrusive. Un site marchand pourrait par exemple (sans vous prévenir) stocker sur votre disque dur vos préférences en matière d'achats et de consultations et de ce fait vous proposer lors d'une session Internet ultérieure des produits et des services correspondant à vos attentes.

En Javascript, un cookie peut être créé en affectant l'objet `document.cookie` avec une chaîne de caractères. Un certain nombre de paramètres sont disponibles pour définir les cookies. Dans le cadre de cet exercice, nous en retiendrons 3 principaux :

- Le nom du cookie. En effet, un même site peut enregistrer autant de cookie qu'il le souhaite.
- Le contenu du cookie. C'est l'élément que l'on veut stocker dans le cookie et récupérer plus tard.
- La date d'expiration. C'est la date à laquelle le cookie expirera. Passé cette date, le cookie n'existera plus.

L'objectif de l'exercice est de stocker le prénom et le nom que l'utilisateur aura saisi dans les champs "Prénom" et "Nom" du formulaire dans un cookie. L'enregistrement du cookie est déclenché lorsque l'utilisateur appuie sur le bouton "Enregistrer". La date d'expiration du cookie doit être de +3 minutes par rapport à la date et l'heure actuelles. Le nom du cookie set "cookie_exercice_13".

Au démarrage de la page, il faut vérifier si un cookie existe. Si c'est le cas, il faut lire le contenu du cookie afin de récupérer le prénom et le nom qui y sont stockés. Une fois ces 2 informations récupérées, il faut les réafficher dans le formulaire.

Inspirez-vous des commentaires laissés dans le code Javascript pour implémenter cette application.

Tests

Saisissez un prénom et un nom dans le formulaire de la page et pressez le bouton "Enregistrer". Fermez la page et ouvrez-la à nouveau pour voir si les données ont bien été récupérées.

Fermez une nouvelle fois la page et attendez 4 à 5 minutes puis vérifiez que les données ne peuvent, cette fois, pas être récupérées. En effet, la date et l'heure d'expiration auront été dépassées.

14.2 Réalisation

D'abord on sauvegarde les données qu'on reçoit dans l'input dans un objet json. En suite on appelle la fonction `setCookie` qui va créer un cookie avec l'objet json et 3 (pour 3min) en paramètres. Ensuite dans la fonction `setCookie` on crée une date d'expiration avec la valeur qu'on reçoit dans le paramètre (fois 60000 parce que ce sont des millisecondes). Après avec « `document.cookies` » on crée notre cookie. Dans la fonction `getCookie` on teste d'abord si le cookie existe (s'il n'est pas égal à 0). Après on prends les valeurs du cookie et on les split dans un objet json. Et à la fin on met la valeur de notre input par rapport l'objet json.

Voici le `main.js` :

```
document.onreadystatechange = function () {
  // Code exécuté lorsque la page a terminé d'être chargée.
  if (document.readyState === "complete") {
    // Chargement de l'éventuelle cookie.
    getCookie();
    // Ajout d'un écouteur "click" sur le bouton "enregistrer".
    document.getElementById("enregistrer").addEventListener("click",
saveData);
  }
};

function saveData() {
  // Récupérer les valeurs contenues dans les champs "prenom" et "nom".
  let prenom = document.getElementById("prenom").value;
  let nom = document.getElementById("nom").value;
  // Créer un objet json "personneJson" contenant le prénom et le "nom".
  let personneJson = {
    prenomJs: prenom,
    nomJs: nom,
  };
  // Convertir l'objet json "personneJson" en chaîne de caractère.
  personneJson = personneJson.prenomJs + " " + personneJson.nomJs;
  // Appeler la méthode "setCookie" en passant en paramètre l'objet
"personneJson" converti et en précisant le délai de 3 minutes.
  setCookie(personneJson, 3);
}

function setCookie(contenu, minutes) {
```

```

// Générer une date d'expiration. Il doit s'agir de la date et l'heure
actuelle + le nombre de minute passé en paramètre.
let exp = new Date().getTime() + (minutes * 60000);

// Créer le cookie.
document.cookie = "cookie_exercice_13=" + contenu + " ; expires=" + exp + "
path="/";
console.log(contenu);
console.log("Data saved in cookie");
}

function getCookie() {
  // Tester la présence du cookie. Pour cela, il faut tester si la taille du
  cookie est supérieure à 0.
  if (document.cookie.length > 0) {
    // Récupérer le contenu du cookie et en extraire la partie qui se situe
    après le "="; Vous pouvez utiliser la méthode "split"
    // liée aux chaînes de caractères.
    let fistTab = document.cookie.split("=")
    let temp = fistTab[1];
    // Convertir la chaîne de caractères en objet Json.
    let tab = temp.split(" ");
    let personneJson = {
      prenom: tab[0],
      nom: tab[1],
    };
    // Charger les deux champs avec la valeur du prénom et du du nom contenu
    dans l'objet json.
    //
    let prenom = document.getElementById("prenom").value =
    personneJson.prenom;
    let nom = document.getElementById("nom").value = personneJson.nom;
    console.log("Data retrieved from cookie");
  }
}

```

Quand on inspecte et on va sur « Application » on peut voir les cookies que le site web a et on peut voir ces propriétés.

The screenshot shows a web browser window with a form titled "Exercice_13/". The form has two input fields: "Prénom :" with the value "jeremy" and "Nom :" with the value "monney". There is an "Enregistrer" button. Below the form, the browser's developer tools are open, specifically the "Application" tab. The "Cookies" section is expanded, showing a table of cookies for the URL "http://127.0.0.1:5500".

Name	Value	Domain	Path	Expires / ...	Size
cookie_exercice_13	jeremy mon...	127.0.0.1	/Exercice_13	Session	31

15 Exercice 14 (Bases de jQuery)

15.1 Description

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.

15.2 Réalisation

jQuery est une bibliothèque JavaScript populaire qui simplifie la manipulation du DOM (Document Object Model) et l'interaction avec les événements. Il fournit une syntaxe concise et puissante pour effectuer des opérations courantes sur les éléments HTML, gérer les animations, effectuer des requêtes AJAX, etc. Le code jQuery est généralement reconnaissable par le signe \$.

15.2.1 Tâches principales de jQuery

Les deux tâches principales de jQuery sont la manipulation du DOM et la gestion des événements. jQuery facilite la sélection et la manipulation des éléments HTML, l'ajout ou la suppression de classes CSS, le changement de contenu, la manipulation des attributs, etc. De plus, jQuery simplifie la gestion des événements comme les clics de souris, les survols etc.

15.2.2 Sélecteurs

Pour trouver des éléments du DOM, on utilise ces sélecteurs :

- Par nom de balise : \$('tagname')
- Par classe CSS : \$('.classname')
- Par ID : \$('#idname')

15.2.3 Ready()

La fonction jQuery appelée quand la page a été chargée est \$(document).ready(). Elle est utilisée pour exécuter du code JavaScript une fois que le DOM de la page est prêt.

Voici un exemple :

```
$(document).ready(function() {
    // Code à exécuter une fois que le DOM est prêt
    // ...
});
```

15.2.4 Héritage

Pour retrouver des éléments parents, suivants, précédents, enfants ou frères avec jQuery, on utilise les fonctions suivantes :

- Parent : `parent()`
- Suivant : `next()`
- Précédent : `prev()`
- Enfant : `children()`
- Éléments frères : `siblings()`

15.2.5 Différentes opérations

Voici comment effectuer différentes opérations avec jQuery :

Modifier le CSS d'un élément sélectionné

```
$(selector).css(property, value)
```

Ajouter un élément avant l'élément sélectionné

```
$(element).insertBefore(target)
```

Ajouter un élément après l'élément sélectionné

```
$(element).insertAfter(target)
```

Supprimer un élément

```
$(element).remove()
```

Gérer les événements

```
$(element).on(event, handler)
```

Cacher un élément

```
$(element).hide()
```

Afficher un élément

```
$(element).show()
```

15.2.6 Lire fichier AJAX, Json

Pour lire un fichier avec AJAX, on utilise \$.ajax() ou \$.get().

Voici un exemple d'utilisation de \$.get() :

```
$.get('file.txt', function(data) {  
    // Le contenu du fichier est disponible dans la variable 'data'  
    console.log(data);  
});
```

Pour lire un fichier JSON avec la méthode \$.getJSON().

Voici un exemple d'utilisation de \$.getJSON() :

```
$.getJSON('file.json', function(data) {  
    // Le contenu du fichier JSON est disponible dans la variable 'data'  
    console.log(data);  
});
```

15.2.7 Charger un fichier HTML

Pour charger un fichier HTML dans un div existant on utilise la méthode load() de jQuery.

Voici un exemple :

```
$('#myDiv').load('content.html');
```

15.2.8 Intégrer jQuery

Pour intégrer jQuery dans des pages, on utilise une balise <script> pour inclure la bibliothèque depuis un CDN ou héberger la bibliothèque localement.

Voici un exemple d'inclusion depuis un CDN :

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

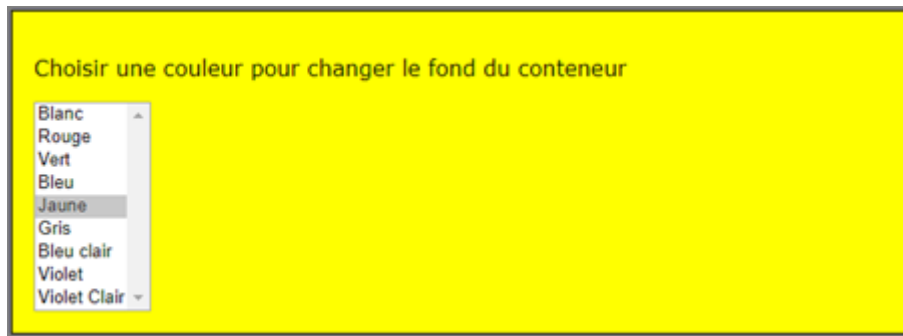
16 Exercice 15 (Première utilisation de jquery)

16.1 Description

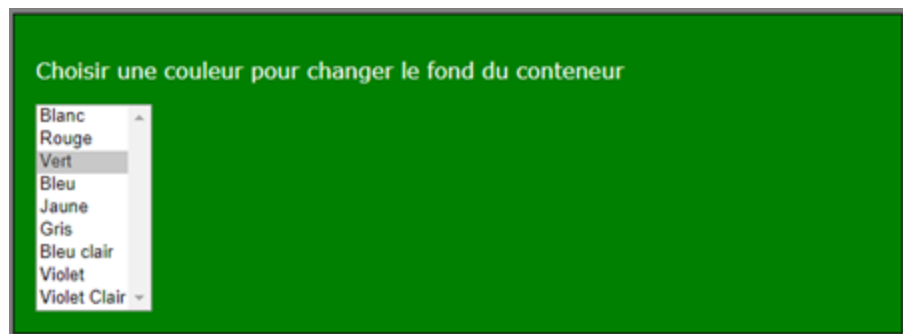
Le but est de créer une vue avec le code HTML ci-dessous, puis de gérer cette vue (changer la couleur du fond du conteneur) avec l'un ou l'autre des 2 contrôleurs proposés (en pur JavaScript ou avec un maximum de JQuery).

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 » :



16.2 Réalisation

On peut voir le `onreadystatechange` qui va exécuter la fonction quand on sélectionne une notre couleur. Et ensuite on voit la classe avec la fonction.

Voici le JS :

```
document.onreadystatechange = function () {
  if (document.readyState === "complete") {
    ctrl = new Ctrl();
    let couleur = document.getElementById("couleurs").value;
    ctrl.changerCouleur(couleur);
  }
};

class Ctrl {
  constructor() {
  }

  changerCouleur(couleur) {
    document.getElementById("container").style.backgroundColor = couleur;
    document.getElementById("titre").style.color = "black";
    if((couleur == "red") || (couleur == "green") || (couleur == "blue")){
      document.getElementById("titre").style.color = "white";
    }
  }
}
```

D'abord on a le `ready()` qui va créer le contrôleur et puis il va directement exécuter la fonction une fois. En suite on a l'écouteur sur `onChange` qui va aussi exécuter la fonction quand on sélectionne une couleur. Et à la fin on peut voir la classe avec la fonction. Pour récupérer la valeur on prends l'ID du élément et on utilise la méthode « `val()` » (`$("#NomDuElement").val()` ;). Pour utiliser les propriétés CSS on peut juste utiliser le « `.css()` » sur l'ID d'un élément. Pour l'animation on utilise `slideToggle` qui dure 1s et 1.5s.

Voici le JQ :

```
$(document).ready(function () {
  ctrl = new Ctrl();
  let couleur = $("#couleurs").val();
  ctrl.changerCouleur(couleur);
});

$("#couleurs").on("change", function () {
  let couleur = $("#couleurs").val();
  ctrl.changerCouleur(couleur);
});

class Ctrl {
  constructor() {}
}
```

```
changerCouleur(couleur) {  
  $("#container").slideToggle(1500, function () {  
    $("#container").css("background-color", couleur);  
    $("#titre").css("color", "black");  
    if (couleur === "red" || couleur === "green" || couleur === "blue") {  
      $("#titre").css("color", "white");  
    }  
    $("#container").slideToggle(1000);  
  });  
}
```

17 Exercice 16

17.1 Description

Dans cet exercice on a dû implémenter plusieurs fonctions qu'avec du JQuery. Chaque bouton a un écouteur qui exécute une fonction.

Exercice 16 - Fonctions jquery

Afficher
Cacher
Afficher / Cacher

Velit sint tempor amet non esse consequat nulla ipsum irure veniam esse commodo veniam eiusmod. Deserunt duis duis esse aute proident proident Lorem. Exercitation non nulla ad minim aliquip esse labore ipsum quis mollit incididunt ipsum. Eu sunt enim do veniam elit cillum magna. Fugiat sit non eiusmod velit officia Lorem ipsum fugiat qui commodo velit veniam. Eiusmod cupidatat non elit proident proident do nisi veniam consectetur elit aliqua incididunt. Incidunt proident consectetur anim labore adipisicing velit commodo ex tempor ea qui voluptate id ullamco.

Plier
Déplier

Sint dolore exercitation sint ex cillum minim velit Lorem id enim anim consequat quis minim. Esse laborum minim veniam ad. Pariatur aliquip occaecat minim laboris. Velit et ullamco non cillum adipisicing ad est non exercitation quis eu labore pariatur fugiat. Nostrud nisi laboris magna amet sit cillum ut ex ex aliquip excepteur Lorem veniam laboris. Et ea irure ea consequat exercitation minim.

Afficher (fondu)
Cacher (fondu)

Culpa incididunt cillum nostrud dolor ad. Magna officia commodo cupidatat in aute eiusmod duis excepteur do duis. Cillum est laborum aute in. Deserunt voluptate anim velit culpa quis ut aute eu nostrud sit consequat in. Cillum nulla culpa dolor id mollit aliquip consequat est cupidatat sunt mollit eiusmod. Incidunt laborum id amet elit ullamco ipsum sunt dolor. Fugiat occaecat cupidatat consectetur adipisicing.

Ajouter une classe au premier élément
Supprimer une classe au premier élément
Ajouter/Supprimer une classe au dernier élément
Souligner les élément impaires

1. Minim do culpa duis dolor cillum enim consectetur pariatur irure anim velit velit.
2. Ullamco aliquip occaecat cillum magna in.
3. Ut aliquip irure sint fugiat nulla ut.
4.
 - o Proident enim ex culpa cupidatat Lorem.
 - o Ad irure nisi deserunt enim incididunt cupidatat sunt.
 - o Eiusmod laboris laborum Lorem esse nisi fugiat enim commodo adipisicing mollit ad excepteur consectetur in.
5. Lorem mollit nulla cillum ea dolor consequat enim ut occaecat proident.

Texte à insérer
☐ Text en gras

José Pascal Henri Sophie

17.2 Réalisation

Les écouteurs des boutons sont tous les mêmes sauf l'écouteur pour l'étape 6. Ils viennent exécuter avec le (document).ready.

Voici les écouteurs :

```
$(document).ready(function () {
    $("#btnShow").on("click", function () {
        afficher();
    });
    $("#btnHide").on("click", function () {
        cacher();
    });
    $("#btnToggle").on("click", function () {
        toggle();
    });
    $("#btnPlier").on("click", function () {
        plier();
    });
    $("#btnDeplier").on("click", function () {
        deplier();
    });
    $("#btnFonduShow").on("click", function () {
        afficherFondu();
    });
    $("#btnFonduHide").on("click", function () {
        cacherFondu();
    });
    $("#btnSetClass").on("click", function () {
        ajouterClass();
    });
    $("#btnRemoveClass").on("click", function () {
        supprimerClass();
    });
    $("#btnToggleClass").on("click", function () {
        toggleClass();
    });
    $("#btnPairesClass").on("click", function () {
        impaire();
    });
    $("#btnInsertText").on("click", function () {
        insereText();
    });
    $(".personne").each(function () {
        $(this).on("click", function () {
            alertKey($(this).attr("pk"));
        });
    });
});
```

Le dernier écouteur fait un foreach sur les personnes. En suite si on click sur un des personnes la fonction alertKey se lance avec l'attribut key de la personne qu'on a cliqué dessus.

Etape 1

La première fonction va afficher le div, la deuxième va le cacher et la troisième le affiche si il est cache et le cache si il est afficher.

```
function afficher() {  
    $("#div_1").show();  
}  
  
let cacher = function () {  
    $("#div_1").hide();  
};  
  
let toggle = () => {  
    $("#div_1").toggle();  
};
```

Etape 2

La première fonction pli le div ca veut dire qui monte avec une durée de 1s et la deuxième va déplier aussi avec une durée de 1s.

```
function plier() {  
    $("#div_2").slideUp(1000);  
}  
  
function deplier() {  
    $("#div_2").slideDown(1000);  
}
```

Etape 3

La première fonction va afficher le div avec une animation fade avec une dure de 1s et la deuxième va se cacher avec l'animation fade.

```
function afficherFondu() {  
    $("#div_3").fadeIn(1000);  
}  
  
function cacherFondu() {  
    $("#div_3").fadeOut(1000);  
}
```

Etape 4

La première fonction va ajouter la classe « boldBlueText » au premier élément de la liste. La deuxième fonction va supprimer la classe. La troisième va l'ajouter si elle n'est pas encore ajoutée et la supprimer si elle est ajoutée. La quatrième va ajouter la classe « underlineClass » à tous les éléments li qui hérite de ol et aux éléments impaires (on utilise :even parce que la liste commence avec 0, 1, 2 ...).

```
function ajouterClass() {
    $("li:first").addClass("boldBlueText");
}

function supprimerClass() {
    $("li:first").removeClass("boldBlueText");
}

function toggleClass() {
    $("li:first").toggleClass("boldBlueText");
}

function impaire() {
    $("ol > li:even").addClass("underlineClass");
}
```

Etape 5

Cette fonction va d'abord tester si la checkbox est cochée ou pas. Si elle est cochée on ajoute la classe « boldBlueText » au div et on ajoute un élément HTML dans ce cas on ajoute un paragraphe et la valeur qu'on obtient dans le input avec la fonction val().

```
function insereText() {
    if ($("#textBold").is(":checked") == true) {
        $("#textes").addClass("boldBlueText");
        $("#textes").append("<p>" + $("#textToInsert").val() + "</p>");
    } else {
        $("#textes").append("<p>" + $("#textToInsert").val() + "</p>");
    }
}
```

Etape 6

Cette fonction va afficher une alert avec la valeur de l'attribut key.

```
function alertKey(key) {
    alert("La clé primaire de la personne cliquée est: " + key);
}
```

18 Exercice 18

18.1 Description

Ci-dessous, deux web services PHP. 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 (JavaScript Object Notation) est plus simple à traiter en JavaScript.

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 JavaScript 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, ...).

Convertisseur de température

°C: °F:

Convertisseur de température

°C: °F:

127.0.0.1:5500 indique
Pas d'accès à la ressource serveur demandée !

OK

18.2 Réalisation

Dans le HTML on fait un form. Quand on lâche une touche la fonction « celsius2Fahrenheit » ce lance.

```
<form class="formulaire">
  <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>
</form>
```

```

    </div>
  </div>
</form>

```

Notre indexCtrl va envoyer les valeurs de l'input au httpServ (degrés). Il gère les erreurs et il affiche les valeurs renvoyées par httpServ (successCallback ou errorCallback).

```

$(document).ready(function () {
  ctrl = new Ctrl();
  httpServ = new HttpServ();
});

class Ctrl {
  constructor() {

  }

  celsius2Fahrenheit() {
    // Lire les degrés du formulaire
    let degres = $("#celsius1").val();
    httpServ.celsius2Fahrenheit(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
    $("#fahrenheit1").val(affiche);
  }

  // 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);
}
});
}
}

```

Notre httpServ va envoyer les données vers le WS et les recevoir et envoyer vers le indexCtrl avec la méthode POST. Le \$ajax va envoyer la requête. SuccessCallback et la valeurs reçu si il y a pas d'erreur et on cas d'erreur errorCallback.

```

class HttpServ {

    constructor(){
    }

    celcius2Fahrenheit(degrees, successCallback, errorCallback) {
        let url = "php/convert-temp_p_xml.php";
        let param = "Temperature=" + degrees + "&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
        });
    }
}

```

Pour simuler une API on a les deux fichiers PHP qui convertisse les degrés en fahrenheit une fois en GET (JSON) et une fois avec POST (XML). Le header du PHP va vérifier l'origine de la requête qui reçoit. Avec « Access-Control-Allow-Origin: * » la requête peut avoir n'importe quelle origine.

XML :

```

<?PHP
// convert_temp_p_xml.php
// Webservice Conversion de température POST et réponse XML
// Naël Telfser
//
header('Content-Type: application/xml');
header('Access-Control-Allow-Origin: *');
$temp = intval($_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;
?>

```

JSON :

```

<?php
// convert_temp_g_json.php
// Webservice Conversion de température GET et réponse JSON
//
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
$temp = intval($_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
?>

```

19 Exercice 20

19.1 Description

On va découper l'application en vue; chaque vue ayant son propre 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 connaître les arguments qu'il faudra transmettre en POST. Il retourne la langue "en, de ou fr" si cela se passe bien, sinon KO.

Maquette

Login avec choix du « *domaine* » en plus

Identification: _____

Nom d'utilisateur:

Mot de passe:

Domaine:

[Créer un compte](#)

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

Nouveau compte

Identification: _____

Nom d'utilisateur:

Mot de passe:

Domaine:

Adresse mail:

Langue: ☒ fr ☐ dt ☐ en

Page après login pour Doe (en anglais)

Good morning Vietnam

Cette page correspond à l'application principale qu'il faudrait développer.

Pour effectuer un logout, cliquez [ici](#).

Trois pages sont chargées 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.

Le projet est 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.

19.2 Réalisation

Dans ce projet on a une page index vide avec un div. Dans ce div on va charger nos autres pages html, c'est le indexCtrl qui va appeler les HTMLs.

Pour faire ça on utilise la classe suivante en JQuery :

```
class VueService {
  constructor() {}

  chargerVue(vue, callback) {

    // charger la vue demandee
    $("#view").load("views/" + vue + ".html", function () {

      // si une fonction de callback est spécifiée, on l'appelle ici
      if (typeof callback !== "undefined") {
        callback();
      }

    });
  }
}
```

Et on voit comment on l'appelle dans le indexCtrl (pour les trois pages login, accueil et compte):

```
// avec arrow function
loadLogin() {
  this.vue.chargerVue("login", () => new LoginCtrl());
}

// avec fonction classique
loadAccueil(langue) {
  this.vue.chargerVue("accueil", function() {
    new AccueilCtrl(langue);
  });
}

loadCompte() {
  this.vue.chargerVue("compte", () => new CompteCtrl());
}
}
```


Le `httpService` va envoyer et recevoir les données. Dans `jQuery`, la méthode `ajaxSetup()` est utilisée pour définir la configuration par défaut de toutes les futures requêtes AJAX. Ici on gère les exceptions. Et le `ajax(url..` va envoyer et recevoir les données.

```
class HttpService {
  constructor() {}

  /**
   ** $.ajaxSetup permet de définir une fois un élément sans le refaire par la
   suite. Ici cela se fait l'error
   */
  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);
      },
    });
  }

  /**
   */
  login(identifiant, successCallback) {
    // Uploade votre propre fichier PHP et adaptez l'URL ci-dessous.
    let url = "https://monneyj.emf-
informatique.ch/Module_307/Exercices/Exercice_20/php/login20.php";
    let param = "username=" + identifiant.username +
      "&password="+identifiant.password + "&domaine=" + identifiant.domaine +
      "&mail="+identifiant.mail+ "&langue="+ identifiant.langue;

    // envoi de la requête
    $.ajax(url, {
      type: "POST",
      contentType: "application/x-www-form-urlencoded; charset=UTF-8",
      data: param,
```

```
    success: successCallback
  });
}
```

20 REST

REST (Representational State Transfer) est pour concevoir des services web. Il utilise les HTTP (GET, POST, PUT, DELETE) pour manipuler les ressources. Il est le plus utilisé pour créer des APIs et des services web modernes.

21 SOAP

SOAP (Simple Object Access Protocol) est un protocole de communication basé sur XML. Il permet d'échanger des informations structurées entre des applications distantes. Les messages SOAP sont enveloppés dans une structure XML et peuvent être transportés via différents protocoles.

22 GET

C'est une méthode HTTP utilisée pour récupérer des données à partir d'un serveur web. Lorsqu'une requête GET est envoyée à un serveur, celui-ci renvoie les données demandées dans la réponse. Les requêtes GET sont généralement utilisées pour lire des ressources sans les modifier.

23 POST

C'est une méthode HTTP utilisée pour envoyer des données à un serveur web pour traitement. Lorsqu'une requête POST est envoyée, les données sont incluses dans le corps de la requête. Le serveur traite les données et renvoie éventuellement une réponse. Les requêtes POST sont souvent utilisées pour créer de nouvelles ressources ou envoyer des informations à un serveur.

24 PUT

C'est une méthode HTTP utilisée pour mettre à jour ou remplacer une ressource existante sur un serveur web. Lorsqu'une requête PUT est envoyée, les données envoyées remplacent la ressource existante. Les requêtes PUT sont utilisées pour modifier des informations existantes.

25 DELETE

C'est une méthode HTTP utilisée pour supprimer une ressource spécifiée sur un serveur web. Lorsqu'une requête DELETE est envoyée, le serveur supprime la ressource demandée. Les requêtes DELETE sont utilisées pour effacer des informations ou des ressources existantes et sont irréversibles.

26 Postman

Postman est une application qui permet de tester des WS comme des APIs on peut faire des GET, POST, PUT, DELETE etc. En haut on peut changer la méthode (GET etc.) et a cote on peut mettre sont url. En suite on peut filtre dans le URL les données qu'on veut et ils sont afficher on bas. Pour voir les informations dans avec POST on doit aller dans le onglet « Body ».

The screenshot shows the Postman interface with a POST request to `https://finnhub.io/api/v1/company-news?symbol=AMZN&from=2023-06-12&to=2023-06-12&token=chusla9r01quokjh2r0chusla9r01qr...`. The request is configured with the following query parameters:

Key	Value	Description
<input checked="" type="checkbox"/> symbol	AMZN	
<input checked="" type="checkbox"/> from	2023-06-12	
<input checked="" type="checkbox"/> to	2023-06-12	
<input checked="" type="checkbox"/> token	chusla9r01quokjh2r0chusla9r01quokjh2rg	
Key	Value	Description

The response is shown in the 'Body' tab, displaying a JSON object with the following structure:

```

1 {
2   "category": "company",
3   "datetime": 1686546740,
4   "headline": "Ahold Delhaize hits 'roughly half' its 1 billion euro goal for non-grocery revenue",
5   "id": 120823877,
6   "image": "https://media.zenfs.com/en/zeuters-finance.com/b73da94ad15d3cfla9e75c644055f375",
7   "related": "AMZN",
8   "source": "Yahoo",
9   "summary": "Ahold Delhaize has hit \"roughly half\" its goal to grow revenue from businesses beyond grocery stores to 1 billion
10  euros by 2025, CEO Frans Muller told Reuters, an effort focused on selling ads on its supermarkets' websites and monetising
11  insights on consumer data. Grocers like Ahold Delhaize are building ventures that sell online advertising to consumer goods
12  companies, including the ability for their brands to appear first in shoppers' searches or feature as \"sponsored products\" -
    offerings pioneered by Amazon. They are also working to improve the quality of the data they can sell to the consumer goods
    companies, who in turn use it to target shoppers with ads for their products.",
13   "url": "https://finnhub.io/api/news?id=dc12f312eb652b12aab51777a21115aadf9959758c7ee8bd1e6548b5f86e87c60"
14 }
  
```

27 Projet

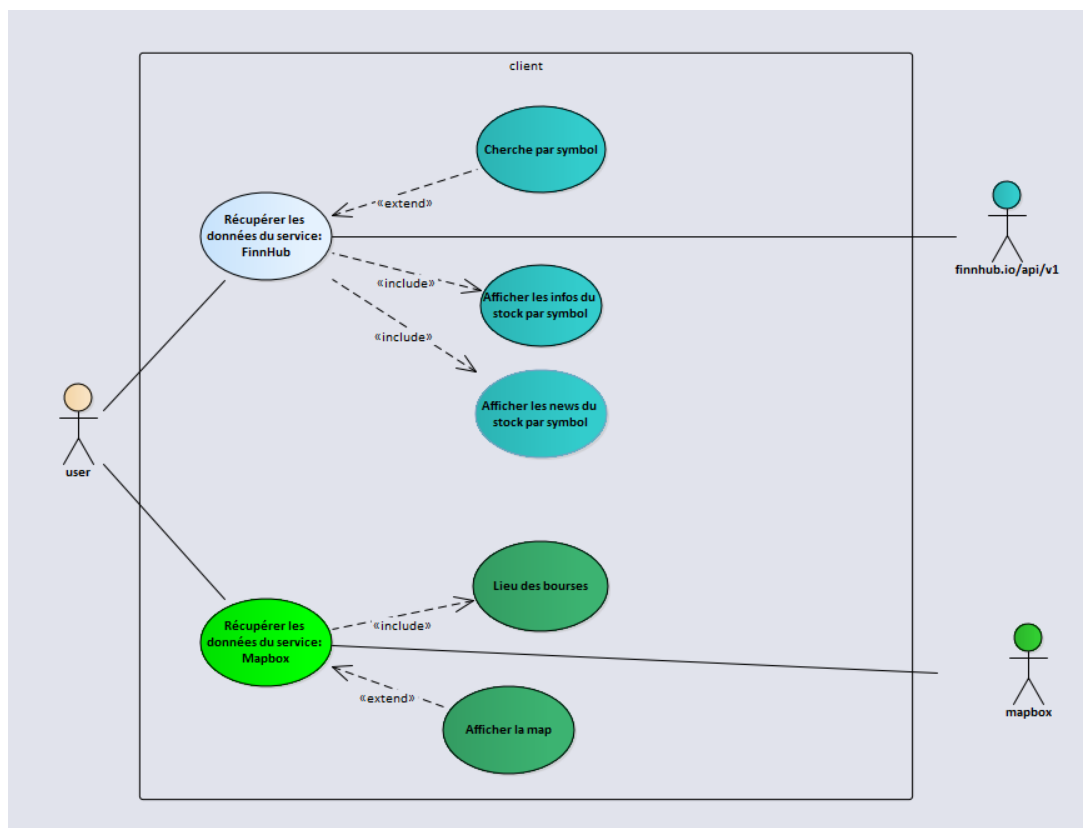
Mon projet s'appelle « American Stock Informations ». On peut obtenir divers informations sur des sociétés d'actions en Amérique. On peut voir un chart qui donne un aperçu du prix de l'entreprise (jusqu'à 1 an en arrière). On peut aussi chercher des nouvelles quotidiennes sur l'entreprise. En plus, il y a une carte qui montre les 10 plus grandes bourses du monde.

American Stock Informations

27.1 Analyse

27.1.1 Diagramme

Voici le diagramme use case :



27.1.2 Maquette

Voici la maquette des différents pages de mon projet :

Index/Homepage :

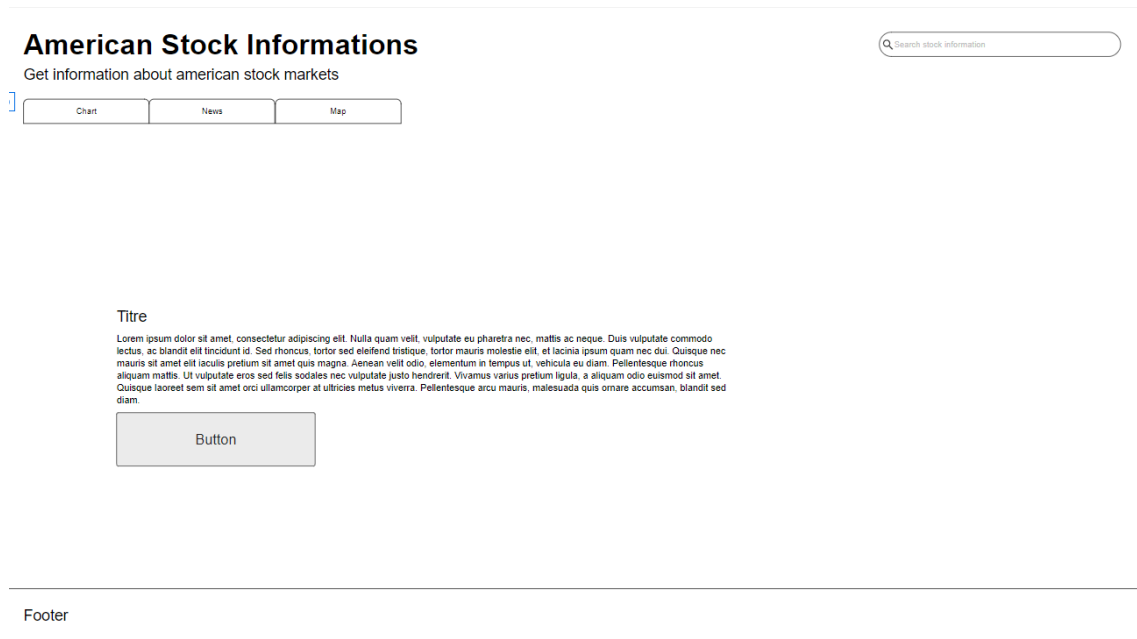
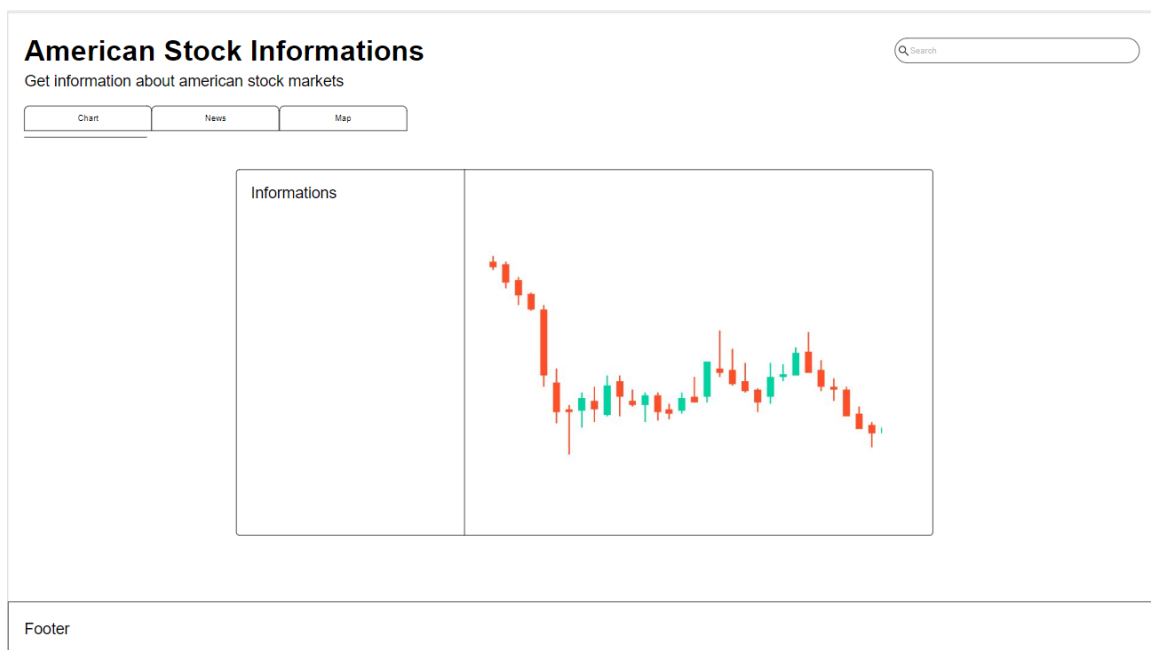


Chart :



307 - Réaliser des pages Web interactives


News :

American Stock Informations

Get information about american stock markets

[Chart](#)[News](#)[Map](#)


Heading
Heading



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit lacus pretium sit amet quia magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed fella sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris.

[link](#)

Heading
Heading



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit lacus pretium sit amet quia magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed fella sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris.

[link](#)

...


Footer

Map :

American Stock Informations

Get information about american stock markets

[Chart](#)[News](#)[Map](#)



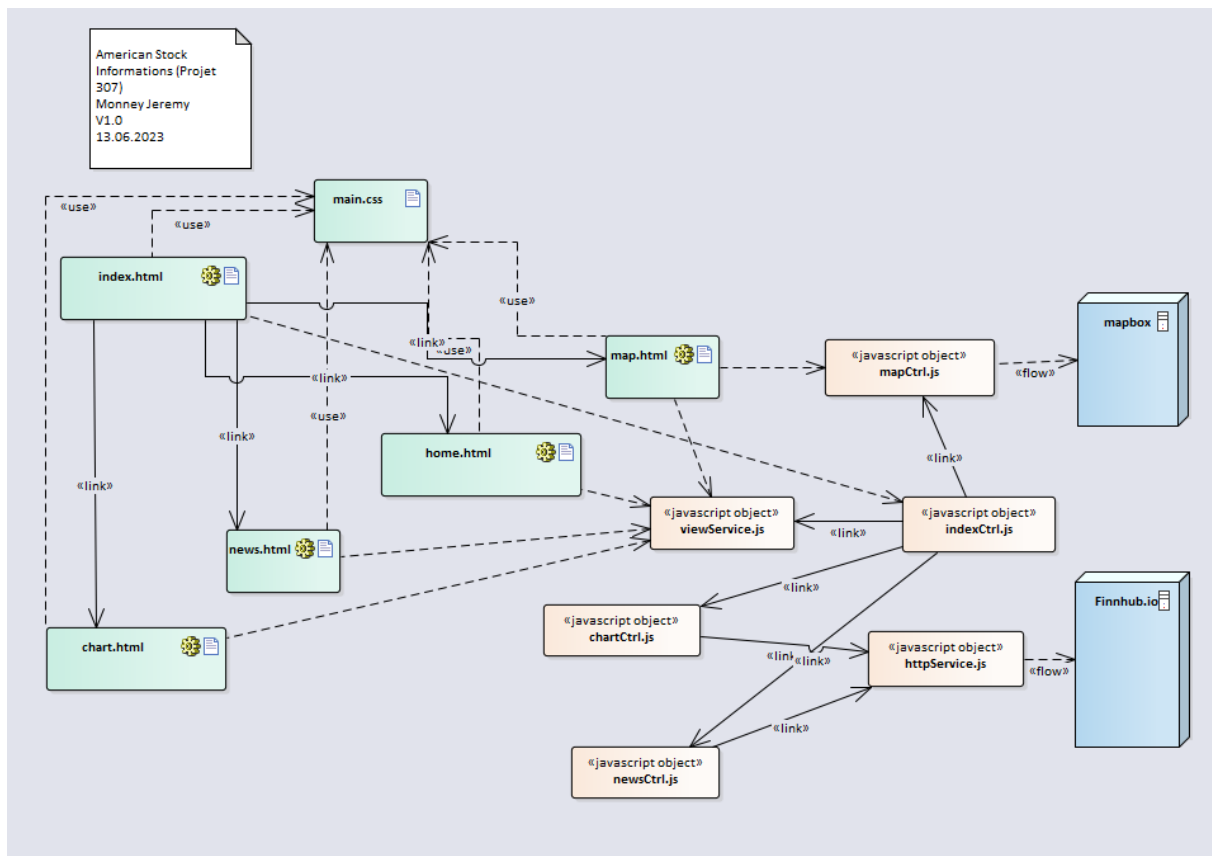
Footer

62 sur 74

27.2 Conception

27.2.1 Diagramme de navigation

Voici le diagramme de navigation :



27.3 Implémentation

27.3.1 HTML

Dans la page Index on contient le Head avec tous les imports pour les fichier JS, JQuery, Highcharts, Mapbox et la liaison avec le stylesheet CSS. On trouve aussi la bar de navigation et le Footer dans le Body. Dans le div « view » on va charger les autres pages HTML.

```

<!DOCTYPE html>
<html>
  <!--
  But :    HTML Index page
  Auteur : Jeremy Monney
  Date :   12.06.2023
  Version: 1.0
  -->

  <!-- entête de la page -->
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" type="image/png" href="images/chart_icon.png" />
  
```

```

<link rel="stylesheet" href="css/main.css" />
<script src="https://code.jquery.com/jquery-3.7.0.min.js"></script>
<script type="text/javascript" src="js/indexCtrl.js"></script>
<script type="text/javascript" src="js/viewService.js"></script>
<script type="text/javascript" src="js/httpService.js"></script>
<script type="text/javascript" src="js/chartCtrl.js"></script>
<script type="text/javascript" src="js/newsCtrl.js"></script>
<script type="text/javascript" src="js/mapCtrl.js"></script>
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/stock.js"></script>
<script src="https://cdn.jsdelivr.net/npm/ol@v7.4.0/dist/ol.js"></script>
<script src="https://api.mapbox.com/mapbox-gl-js/v2.15.0/mapbox-
gl.js"></script>
<link
  href="https://api.mapbox.com/mapbox-gl-js/v2.15.0/mapbox-gl.css"
  rel="stylesheet"
/>
<link
  href="https://fonts.googleapis.com/css?family=Epilogue"
  rel="stylesheet"
/>
<title>American Stock Informations</title>
</head>

<!-- corps de la page -->
<body>
  <header>
    <div class="header-container">
      <div>
        <a class="link_index" id="id_home" onclick="indexCtrl.loadHome()"
          ><h1>American Stock Informations</h1></a>
      </div>
      <!--Search bar-->
      <div class="search-bar-container">
        <input
          class="search-bar"
          id="search-bar"
          type="text"
          placeholder="Search symbol..."
        />
      </div>
    </div>
    <h2>Get reliable information about american stock markets</h2>
  </header>
  <!--Bar de navigation-->
  <nav>
    <ul>
      <li>

```



```

        <button id="id_chart" class="navbtn"
onclick="indexCtrl.loadChart()">
            Chart
        </button>
    </li>
    <li>
        <button id="id_news" class="navbtn" onclick="indexCtrl.loadNews()">
            News
        </button>
    </li>
    <li>
        <button id="id_map" class="navbtn" onclick="indexCtrl.loadMap()">
            Map
        </button>
    </li>
</ul>
</nav>
<div id="container" class="container">
    <!--container for the different views -->
    <div id="view" class="view"></div>
</div>
<!-- end -->
<footer>
    <p>Made by Monney Jeremy © Project 307</p>
</footer>
</body>
</html>

```

Les autres HTML contient presque que des DIV ou des titres. Le contenu va être chargé par le viewService et le IndexCtrl.

27.3.2 CSS

Le CSS contient pas de particularité spéciales. Il y a deux media queries pour un petit écran (max-width : 576px) et pour des moyen grand écrans (min-width : 576px) et (max-width: 992px).

27.3.3 JS

27.3.3.1 httpService et viewService

Le httpService.js va gérer tous les appels aux APIs de Finnhub.io.

centraliserErreurHttp

Cette fonction va permettre de gérer les erreurs éventuellement rencontrées par l'API.

getQuote

Cette fonction va permettre d'obtenir les informations qui se trouvent à côté du chart. Avec le paramètre de Symbol on peut obtenir les données de Finnhub. L'appel AJAX va rendre les données avec le url qui contient le filtre (symbole) et le token.

getChartData

Cette fonction va permettre d'obtenir les données pour créer le chart. D'abord on crée deux variables qui contiennent les dates d'aujourd'hui (toTimestamp) et exactement une année avant (fromTimestamp) parce que l'API retourne que des valeurs d'une année. L'appel AJAX va rendre les datas avec le url qui contient le filtre (symbole).

getNews

Cette fonction va permettre d'obtenir les données pour les news. D'abord on crée une variable qui contient la date d'aujourd'hui dans le format 0000-00-00. Et par rapport à la date et le symbol on reçoit après les news d'aujourd'hui.

L'appel AJAX va rendre les datas avec le url qui contient le filtre (symbole) et le token.

```
class HttpService {
  constructor() {}
  //Error management
  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);
      },
    });
  }
  //Gets chart infos
  getQuote(symbol, successCallback) {
    let url =
      "https://finnhub.io/api/v1/quote?symbol=" +
      symbol +
      "&token=chusla9r01quokjh2r0chusla9r01quokjh2rg";

    $.ajax(url, {
      type: "GET",
      contentType: "application/x-www-form-urlencoded; charset=UTF-8",
      success: successCallback,
    });
  }
}
```

```

//Gets data for chart
getChartData(symbol, successCallBack) {
  let currentDate = new Date();
  let oneYearAgo = new Date();
  oneYearAgo.setFullYear(currentDate.getFullYear() - 1);

  let fromTimestamp = Math.floor(oneYearAgo.getTime() / 1000);
  let toTimestamp = Math.floor(currentDate.getTime() / 1000);

  return $.ajax({
    url: "https://finnhub.io/api/v1/stock/candle",
    method: "GET",
    success: successCallBack,
    data: {
      symbol: symbol,
      resolution: "D",
      from: fromTimestamp,
      to: toTimestamp,
      token: "chusla9r01quokjh2r0chusla9r01quokjh2rg",
    },
  });
}

//Gets news infos
getNews(symbol, successCallBack) {
  let currentDate = new Date();
  let year = currentDate.getFullYear();
  let month = String(currentDate.getMonth() + 1).padStart(2, "0");
  let day = String(currentDate.getDate()).padStart(2, "0");
  let formattedDate = year + "-" + month + "-" + day;
  let url =
    "https://finnhub.io/api/v1/company-news?symbol=" +
    symbol +
    "&from=" +
    formattedDate +
    "&to=" +
    formattedDate +
    "&token=chusla9r01quokjh2r0chusla9r01quokjh2rg";
  $.ajax(url, {
    type: "GET",
    contentType: "application/x-www-form-urlencoded; charset=UTF-8",
    success: successCallBack,
  });
}
}

```

ViewService va permettre au IndexCtrl de charger les pages HTML dans l'index. On prend le nom du HTML qui viens appeler dans le paramètres « vue » et on le charge avec du JQuery dans le div avec l'ID « view ».

```
class VueService {
  constructor() {}
  //to load the different htmls
  chargerVue(vue, callback) {
    $("#view").load("views/" + vue + ".html", function () {
      if (typeof callback !== "undefined") {
        callback();
      }
    });
  }

  chargerVueSimple(vue) {
    $("#view").load("views/" + vue + ".html");
  }
}
```

27.3.4 Controller

Chaque page contient un contrôler.

indexCtrl

D'abord on crée l'objet indexCtrl quand le document est prêts avec le JQuery .ready. Dans le constructeur on crée l'objet vue pour charger les HTML. Après on a la fonction afficherErreurHTTP qui va afficher un message dans le cas d'erreur. A la fin on a les différents load qui vont charger les HTML et crée les contrôleurs.

```
$(document).ready(function () {
  indexCtrl = new IndexCtrl();
});

class IndexCtrl {
  constructor() {
    this.vue = new ViewService();
    this.loadHome();
  }

  afficherErreurHttp(msg) {
    alert(msg);
  }

  //Loads the htmls
  loadHome() {
    this.vue.chargerVueSimple("home");
  }

  loadChart() {
    this.vue.chargerVue("chart", () => new ChartCtrl());
  }
}
```

```

loadNews() {
  this.vue.chargerVue("news", () => new NewsCtrl());
}

loadMap() {
  this.vue.chargerVue("map", () => new MapCtrl());
}
}

```

chartCtrl

Le showQuote va afficher les différentes informations à côté du chart. Avec le test on regarde si on reçoit les données ou pas (si le symbol existe ou pas). Et avec un innerHTML on les affiche dans les différents paragraphes.

Finnhub doc :

<https://finnhub.io/docs/api/quote>

Le showChart va créer le chart avec l'aide de la bibliothèque Highcharts. D'abord il attend jusqu'à ce que toutes les données soient chargées avec le done(). Le while va d'abord clear le chart avant de le créer. Ensuite on teste si on reçoit des données ou pas. Le for va après insérer toutes les données dans un tableau qu'on utilise après dans le Highcharts.stockChart pour afficher le chart.

Voici la doc de Highcharts :

https://api.highcharts.com/highstock/plotOptions.candlestick?_gl=1*1fijbno*_up*MQ..&gclid=CjwKCAjwp6CkBhB_EiwAIQVyxWlc8PzY1yJwFvL8deZ6Qp_FmOjE-NF0U0OvwrXaAeg979_8L0z3lXoCivQQAxD_BwE

Et voici la doc de Finnhub pour comprendre les données du chart :

<https://finnhub.io/docs/api/stock-candles>

```

class ChartCtrl {
  constructor() {
    this.httpServ = new HttpService();
    this.showQuote("AAPL");
    this.showChart("AAPL");

    const searchBar = document.getElementById("search-bar");
    const self = this;
    //listener on pressing ENTER
    searchBar.addEventListener("keyup", function (event) {
      if (event.keyCode === 13) {
        const searchVal = searchBar.value;
        if (searchVal === "") {
        } else {
          self.showQuote(searchVal);
          self.showChart(searchVal);
        }
      }
    });
  }
}

```

```

}
//Shows the chart infos in the html
//data from httpservice
showQuote(symbol) {
  this.httpServ.getQuote(symbol, (data) => {
    if (data.c === 0) {
      document.getElementById("chart_div2").innerHTML =
        "<p class='detail_p' style='color: red;'>Invalid symbol!!!</p>";
    }else{
      document.getElementById("chart_symbol").innerHTML = symbol;
      document.getElementById("chart_c").innerHTML = data.c;
      document.getElementById("chart_d").innerHTML = data.d;
      if (data.dp > 0) {
        document.getElementById("chart_dp").innerHTML = data.dp + "%";
        document.getElementById("chart_dp").style.color = "#2bd900";
      } else {
        document.getElementById("chart_dp").innerHTML = data.dp + "%";
        document.getElementById("chart_dp").style.color = "red";
      }
      document.getElementById("chart_h").innerHTML = data.h;
      document.getElementById("chart_l").innerHTML = data.l;
      document.getElementById("chart_o").innerHTML = data.o;
      document.getElementById("chart_pc").innerHTML = data.pc;
    }
  });
}
//Shows the chart in the html
//data from httpservice
showChart(symbol) {
  const chartContainer = document.getElementById("chart-container");
  // Destroy the existing chart
  while (chartContainer.firstChild) {
    chartContainer.firstChild.remove();
  }
  this.httpServ.getChartData(symbol).done(function (response) {
    let seriesData = [];
    if(response.t === undefined){
      console.log("No data");
    }else{
      for (var i = 0; i < response.t.length; i++) {
        seriesData.push([
          response.t[i] * 1000,
          response.o[i],
          response.h[i],
          response.l[i],
          response.c[i],
        ]);
      }
    }
    Highcharts.stockChart("chart-container", {

```

```

        series: [
            {
                type: "candlestick",
                data: seriesData,
            },
        ],
        // Additional chart options
    });
}
});
}
}

```

newsCtrl

Le showNews va afficher tous les news. D'abord on teste si l'objet Json qu'on reçoit est null ou pas (`Object.keys(data).length === 0`). En suite on fait une boucle for et on crée plusieurs éléments HTML avec `createElement`, `setAttribute` pour mettre des attribut a ces éléments HTML et `appendChild` qui va le toujours ajouter dans le div. On crée les éléments HTML dans le JS parce que on a plusieurs news qui utilisent à chaque fois un nouveau div etc.

Voici la doc Finnhubv pour les news :

<https://finnhub.io/docs/api/company-news>

```

class NewsCtrl {
  constructor() {
    this.httpServ = new HttpService();
    this.showNews("AAPL");

    const searchBar = document.getElementById("search-bar");
    const self = this;
    //listener on pressing ENTER
    searchBar.addEventListener("keyup", function (event) {
      if (event.keyCode === 13) {
        const searchVal = searchBar.value;
        if (searchVal === "") {
        } else {
          self.showNews(searchVal);
        }
      }
    });
  }
  //Shows the news in the html
  //data from httpservice
  showNews(symbol) {
    document.getElementById("search-bar").innerHTML = "";
    document.getElementById("newsContainer").innerHTML = "";
    this.httpServ.getNews(symbol, (data) => {
      if (Object.keys(data).length === 0) {
        document.getElementById("newsContainer").innerHTML =

```

```

        "<p class='detail_p' style='color: red;'>No news or Invalid
symbol!!!</p>";
    } else {
        for (let i = 0; i < data.length; i++) {
            const newsDiv = document.createElement("div");
            newsDiv.classList.add("news-item");

            if (data[i].image == "") {
                const imageElement = document.createElement("img");
                imageElement.setAttribute("src", "/images/placeholder_news.jpg");
                newsDiv.appendChild(imageElement);
            } else {
                const imageElement = document.createElement("img");
                imageElement.setAttribute("src", data[i].image);
                newsDiv.appendChild(imageElement);
            }

            const contentDiv = document.createElement("div");
            contentDiv.classList.add("content");

            const headlineElement = document.createElement("h2");
            headlineElement.innerHTML = data[i].headline;
            contentDiv.appendChild(headlineElement);

            const summaryElement = document.createElement("p");
            summaryElement.innerHTML = data[i].summary;
            contentDiv.appendChild(summaryElement);

            const sourceElement = document.createElement("p");
            sourceElement.innerHTML = "Source: " + data[i].source;
            contentDiv.appendChild(sourceElement);

            const linkElement = document.createElement("a");
            linkElement.setAttribute("href", data[i].url);
            linkElement.innerHTML = "Read more";
            contentDiv.appendChild(linkElement);

            newsDiv.appendChild(contentDiv);

            const container = document.getElementById("newsContainer");
            container.appendChild(newsDiv);
        }
    }
});
}
}

```


mapCtrl

Dans le `initMap` on va appeler la map et l'afficher dans le div « map ». La variable `locations` contient les coordonnées des 10 plus grands bourses au monde qui permettent de ensuite être utilise pour crée des points sur la map. Pour crée les points sur la map on utilise `setLngLat()` et le `setPopUp()` va permettre de montre les noms de la bourse quand on clique sur le point.

```
class MapCtrl {
  constructor() {
    this.initMap();
  }
  //Show the map
  initMap() {
    mapboxgl.accessToken =
      "pk.eyJ1IjoibW9ubmV5ajA2IiwiaYSI6ImNsaXN4ZDM3NjE5N2szZG54bGI5MTZzY3oifQ.L
Z9qpMs03z07gD2Co_Sfiw";
    const map = new mapboxgl.Map({
      container: "map",
      style: "mapbox://styles/mapbox/streets-v12",
      center: [-74.009, 40.705],
      zoom: 9,
    });

    // Locations
    let locations = [
      { name: "New York Stock Exchange", coordinates: [-74.009, 40.705] },
      { name: "NASDAQ", coordinates: [-74.037, 40.7128] },
      { name: "Tokyo Stock Exchange", coordinates: [139.6917, 35.6895] },
      { name: "Shanghai Stock Exchange", coordinates: [121.4905, 31.2222] },
      { name: "Hong Kong Stock Exchange", coordinates: [114.1667, 22.2793] },
      { name: "London Stock Exchange", coordinates: [-0.095, 51.5151] },
      { name: "Euronext Stock Exchange", coordinates: [4.8994, 52.3792] },
      { name: "Shenzhen Stock Exchange", coordinates: [114.0579, 22.5431] },
      { name: "Toronto Stock Exchange", coordinates: [-79.3832, 43.6511] },
      { name: "Frankfurter Börse", coordinates: [8.6638, 50.1109] },
    ];

    // Add markers to the map
    locations.forEach(function (location) {
      let marker = new mapboxgl.Marker()
        .setLngLat(location.coordinates)
        .setPopup(
          new mapboxgl.Popup().setHTML("<strong>" + location.name +
"</strong>")
        )
        .addTo(map);
    });
  }
}
```

27.4 Tests

Voici les tests :

Page/GET	Résultat
Chart infos	Ok
Chart	Ok
News	Ok
Map	Ok

27.5 Hébergement et fonctionnement

Le siteweb est hébergé avec CPanel dans public_html et sur GitHub.

Voici le lien :

<https://monneyj.emf-informatique.ch/Exercices/Projet/Projet307/index.html>

Voici le lien GitHub :

<https://github.com/jeremymonney/Projet307.git>

Les APIs utilisées sont :

<https://finnhub.io>

<https://www.mapbox.com>

28 Conclusion

Ce module était très intéressant. J'aime beaucoup le développement web et comme nous n'en faisons pas beaucoup, j'ai trouvé très intéressant de travailler avec JavaScript et les Webservices. Je pense qu'il faudrait aller un peu plus en profondeur dans le JS, car il est tellement utilisé ces jours, mais ce n'est presque pas possible dans un module.

Le projet était aussi très cool à réaliser. Le temps était un peu court, mais c'était encore possible à la fin.