

# 307 - Réaliser des pages Web interactives

Rapport personnel
-------------------

Date de création : 15.05.2023

Version 1 du : 15.06.2023

Carrard Rémi

# Table des matières

---

<b>1</b>	<b>Introduction .....</b>	<b>5</b>
<b>2</b>	<b>Exercice 1 (Découverte des écouteurs).....</b>	<b>5</b>
<b>3</b>	<b>Exercice 2 .....</b>	<b>6</b>
3.1	Objectif de l'exercice .....	6
3.2	Explications / Description .....	6
3.3	Extrait de code .....	7
3.4	Capture .....	7
<b>4</b>	<b>Exercice 3 (PHP).....</b>	<b>8</b>
4.1	Objectif de l'exercice .....	8
4.2	Explications / Description .....	8
4.3	Extrait de code .....	9
4.4	Capture .....	9
<b>5</b>	<b>Exercice 4 (JSFiddle) .....</b>	<b>9</b>
5.1	Objectif de l'exercice .....	9
5.2	Explications / Description .....	9
5.3	Extrait de code .....	10
5.4	Capture .....	10
<b>6</b>	<b>Exercice 5 .....</b>	<b>10</b>
6.1	Objectif de l'exercice .....	10
6.2	Explications / Description .....	10
6.3	Extrait de code .....	11
6.4	Capture .....	12
<b>7</b>	<b>Exercice 6 .....</b>	<b>13</b>
7.1	Objectif de l'exercice .....	13
7.2	Explications / Description .....	13
7.3	Extrait de code .....	14
7.4	Capture .....	14
<b>8</b>	<b>Exercice 7 .....</b>	<b>15</b>
8.1	Objectif de l'exercice .....	15
8.2	Explications / Description .....	15
8.3	Extrait de code .....	15
8.4	Capture .....	15
<b>9</b>	<b>Exercice 8 .....</b>	<b>16</b>
9.1	Objectif de l'exercice .....	16
9.2	Explications / Description .....	16

9.3	Extrait de code .....	16
9.4	Capture .....	17
<b>10</b>	<b>Exercice 9.....</b>	<b>17</b>
10.1	Objectif de l'exercice .....	17
10.2	Explications / Description .....	17
10.3	Extrait de code .....	17
10.4	Capture .....	18
<b>11</b>	<b>Exercice 10.....</b>	<b>18</b>
11.1	Objectif de l'exercice .....	18
11.2	Explications / Description .....	18
11.3	Extrait de code .....	19
11.4	Capture .....	20
<b>12</b>	<b>Exercice 11.....</b>	<b>20</b>
12.1	Objectif de l'exercice .....	20
12.2	Explications / Description .....	20
12.3	Extrait de code .....	21
12.4	Capture .....	21
<b>13</b>	<b>Exercice 12.....</b>	<b>22</b>
13.1	Objectif de l'exercice .....	22
13.2	Explications / Description .....	22
13.3	Extrait de code .....	23
13.4	Capture .....	23
<b>14</b>	<b>Exercice 13.....</b>	<b>23</b>
14.1	Objectif de l'exercice .....	23
14.2	Explications / Description .....	23
14.3	Extrait de code .....	24
14.4	Capture .....	24
<b>15</b>	<b>Exercice 14.....</b>	<b>25</b>
15.1	Objectif de l'exercice .....	25
15.2	Explications / Description .....	25
15.3	Extrait de code .....	27
15.4	Capture .....	27
<b>16</b>	<b>Exercice 15.....</b>	<b>28</b>
16.1	Objectif de l'exercice .....	28
16.2	Explications / Description .....	28
16.3	Extrait de code .....	28
16.4	Capture .....	29

<b>17</b>	<b>Exercice 16</b>	<b>29</b>
17.1	Objectif de l'exercice	29
17.2	Explications / Description	29
17.3	Extrait de code	30
17.4	Capture	31
<b>18</b>	<b>Exercice 17</b>	<b>31</b>
18.1	Objectif de l'exercice	31
18.2	Explications / Description	31
18.2.1	REST	31
18.2.2	SOAP	32
<b>19</b>	<b>Exercice 18</b>	<b>32</b>
19.1	Objectif de l'exercice	32
19.2	Explications / Description	32
19.3	Extrait de code	33
19.4	Capture	34
<b>20</b>	<b>Exercice 20</b>	<b>35</b>
20.1	Objectif de l'exercice	35
20.2	Explications / Description	35
20.3	Extrait de code	35
20.4	Capture	36
<b>21</b>	<b>Projet</b>	<b>37</b>
21.1	Analyse	37
21.1.1	Diagramme de use cases / Explications des cas	38
21.1.2	Maquette	39
21.2	Conception	41
21.2.1	Diagramme de navigation	41
21.3	Implémentation	41
21.3.1	HTML / CSS	42
21.3.2	Javascript	43
21.3.3	Appel au webService	43
21.4	Tests	44
21.4.1	Postman	44
21.4.2	Fonctionnement sur le site	45
21.5	Hébergement et fonctionnement	45
<b>22</b>	<b>Conclusion</b>	<b>45</b>

# 1 Introduction

Ce module est un module de programmation et plus précisément de développement web. On va principalement découvrir javascript avec également la librairie JQuery et notamment les appels ajax. Voici les objectifs du module :

- 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 (Découverte des écouteurs)

**Comment ajouter un écouteur "click" via Javascript.**

Lorsque l'on lance l'application, on va appeler une méthode pour ajouter un écouteur. Voici la ligne de code que l'on va mettre dans la méthode dans le javascript :

```
document.getElementById("testez").addEventListener("click", testez);
```

Le premier « testez » correspond à l'élément sur lequel appliquer l'écouteur, ensuite le « click » correspond au type d'action auxquels l'écouteur va réagir. Et le dernier testez est la méthode que l'on va faire lorsque l'écouteur s'activera.

Et maintenant sur notre élément où l'on veut ajouter l'écouteur, on va ajouter un id avec le nom testez :

```
<button id="testez">
```

**Comment ajouter directement un écouteur "click" sur un bouton.**

Pour ajouter l'écouteur directement sur le bouton, on va ajouter un attribut onclick avec le nom de la méthode à exécuter lorsque l'écouteur sera activé. Voici un exemple :

```
<button onclick="testez()">
```

**Expliquez comment exécuter du code Javascript lorsque la page HTML a fini d'être chargée.**

Pour pouvoir exécuter une méthode lorsque la page aura fini de se charger, on va modifier le body et mettre cela :

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

Le initCtrl est le nom de la méthode à lancer.

**Expliquez ce que signifie DOM.**

Il signifie Document Object Model et permet de créer un modèle pour javascript qui converti tous les éléments en object pour que l'on puisse par la suite effectuer des requêtes avec javascript sur notre document html.

**Où le script Javascript peuvent être chargés et pourquoi.**

Il est chargé dans le body et dans le onload. Il est exécuté à ce moment étant donné que si on le fait avant, le script sera exécuté alors que les éléments ne sont pas encore créés.

## 3 Exercice 2

### 3.1 Objectif de l'exercice

Dans cet exercice, le but sera de découvrir un peu les débuts de javascript. Le but sera de contrôler si un username et un password écrit par un utilisateur correspond à des valeurs en dur dans le programme.

### 3.2 Explications / Description

Le code pour le fichier html était donné et pour le css et le javascript, nous avons dû le faire nous-même.

Tout d'abord, il y aura les fonction alert(), confirm() et prompt() qui permettent de faire des popup. Alert() donne une information, confirm() une popup de confirmation et prompt() demande du texte.

L'attribut placeholder dans les textes d'inputs permettent de mettre une description de ce que l'on veut dans le champ. Et l'attribut autofocus permet de spécifier que ce champ sera mis en avant au lancement de la page.

La différence entre une balise button et un bouton input est que pour un input, on ne pourra mettre que du texte tandis que dans un button, on peut mettre une image.

Ensuite pour récupérer du texte dans notre input de texte par exemple, on va d'abord récupérer l'élément grâce à document.getElementById et ensuite on va mettre un point et value. Voici un exemple :

```
document.getElementById("username").value
```

Comme dans netbeans, on peut écrire dans la console. Pour cela, on va faire :

```
Console.log("texte à écrire")
```

Cela permet par exemple de voir une valeur d'un élément à un moment donné. Le message sera donc affiché dans la console et pour le voir, on va faire « inspecter la page » et ensuite on va sous la console. Cela nous permet aussi de voir les différentes erreurs.

Pour créer une fonction, on va juste mettre function et le nom de la fonction. Voici un exemple :

```
function validerUtilisateur() {}
```

Et pour créer une variable, on va faire comme dans java mais on ne met pas le type au début mais « let » voici un exemple :

```
let username = "admin"
```

Les variables n'ont donc pas de type.

Pour convertir une chaîne de caractère en minuscule, on va faire la chaîne de caractère point `toLowerCase()`. Voici un exemple :

```
username = username.toLowerCase()
```

### 3.3 Extrait de code

Voici la fonction qui va être effectuée lorsque l'on va appuyer sur le bouton valider :

```
function validerUtilisateur() {
  let username = document.getElementById("username").value
  let password = document.getElementById("password").value
  username = username.toLowerCase()
  console.log(username)
  console.log(password)
  if (username == "admin" && password == "emf123") {
    window.alert("Validation OK.")
  } else {
    window.alert("Utilisateur ou mot de passe incorrect !!!")
  }
}
```

### 3.4 Capture

Voici tout d'abord une capture du site au début quand l'utilisateur n'a rien écrit :

Et maintenant lorsque l'on fait le login juste :

Et pour finir lorsque le login n'est pas juste :

Identification: —

Nom d'utilisateur:

Mot de passe:

Valider

carrandr.emf-informatique.ch indique

Utilisateur ou mot de passe incorrect !!!

OK

## 4 Exercice 3 (PHP)

### 4.1 Objectif de l'exercice

Dans cet exercice, le but sera de découvrir les liens avec le backend et frontend.

### 4.2 Explications / Description

Dans cet exercice, on va donc reprendre le même exercice que le 2 mais à la place de faire le code dans javascript, on va le faire dans PHP.

Voici la liste des boutons disponible avec une petite explication :

- Button : Simple bouton cliquable
- Submit : Bouton qui envoie un formulaire vers un serveur
- Reset : Réinitialiser les valeurs d'un formulaire
- Image : Bouton représenté par une image
- Checkbox : Bouton qui peut être coché ou décoché
- Radio : Bouton sélectionné dans un groupe de bouton
- File : Bouton où l'on peut sélectionner un fichier par le client

Un fichier PHP va être de cette manière :

```
<?PHP
?>
```

Ensuite, on devra choisir si on veut faire un GET ou un POST étant donné que de base pour récupérer des informations, on va utiliser le GET mais pour un login, on va plus souvent utiliser un POST parce que comme ça les paramètres ne seront pas affichés dans l'URL. Mais le type de requête que l'on va définir dans l'html devra être le même que celui utilisé dans PHP.

Pour spécifier les éléments que l'on va envoyer au fichier PHP, dans les input, on va ajouter un attribut name avec le nom qu'on veut qu'il soit envoyé.

Pour concaténer deux chaînes de caractères, nous allons utiliser le point. Voici un exemple :

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

Ensuite pour mettre en minuscule, on va utiliser la fonction strtolower. Voici un exemple :

```
$username = strtolower($_POST['username']);
```



Pour renvoyer des éléments au client, on va utiliser le echo. Voici un exemple où l'on renvoie quelque chose.

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

Dans notre fichier PHP, on va aussi retrouver du javascript.

### 4.3 Extrait de code

Pour les extraits de code, il n'y en a pas beaucoup. Mais voici quand même le code pour le bouton :

```
<div class="button">
    <input type="submit" value="Valider" id="valider">
</div>
```

Ensuite voici un exemple de l'input pour le username :

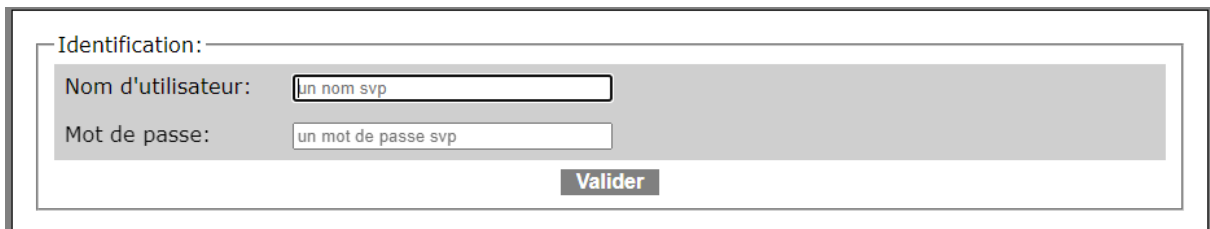
```
<input type="text" size="30" id="username"
placeholder="un nom svp" autofocus name="username"/>
```

Et pour finir, voici la ligne pour le form :

```
<form class="user-form" action="https://carrardr.emf-
informatique.ch/307/Exercices/Exercice_3/login.php" method="post">
```

### 4.4 Capture

Pour les captures d'écran, cela sera les mêmes que pour l'exercice d'avant sauf que le design a un peu changé donc voici le nouveau design :



## 5 Exercice 4 (JSFiddle)

### 5.1 Objectif de l'exercice

Dans cet exercice, l'objectif est de découvrir le site JSFiddle et de voir à quoi il sert.

### 5.2 Explications / Description

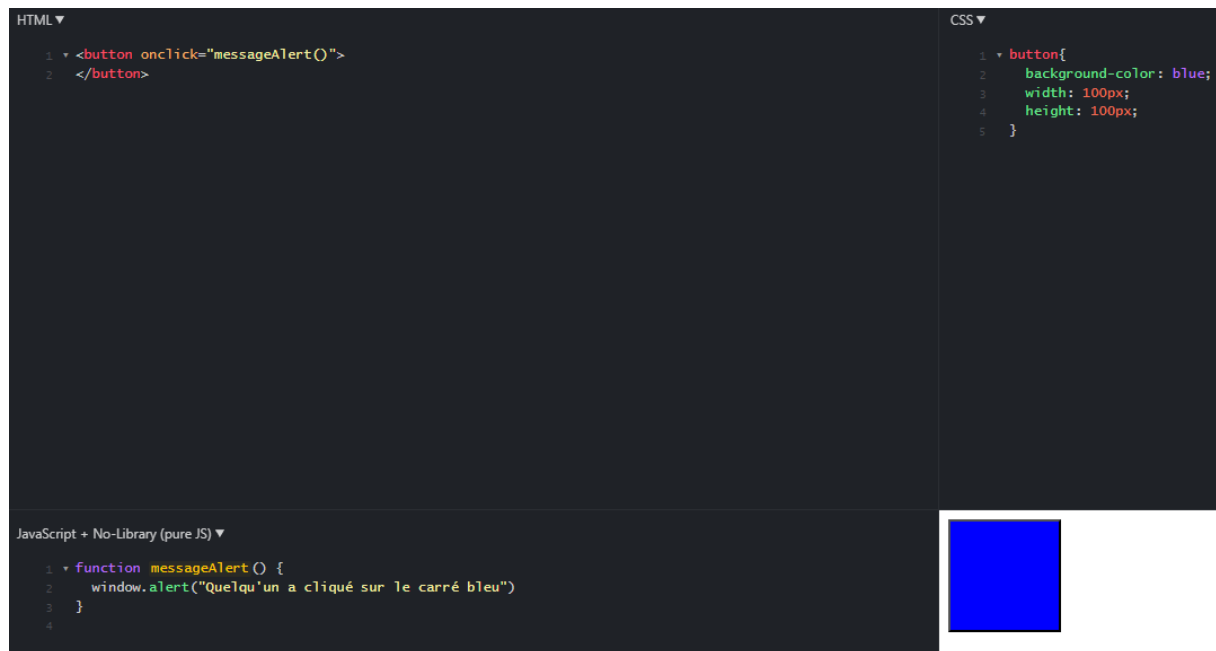
JSFiddle est un site disponible sur internet qui permet de tester du code HTML, CSS et javascript. Cela nous permet donc de voir si notre code fonctionne et si tout est comme on le veut. L'avantage est que l'on n'a pas besoin de faire toute la structure de base pour html. On peut directement commencer par une balise avec un bouton par exemple.

## 5.3 Extrait de code

Pour cet exercice, il n'y aura pas d'extrait de code étant donné que le code que l'on doit faire est le même que celui dans les exercices précédents.

## 5.4 Capture

Et pour la capture, voici la capture d'écran du site en question avec le résultat de l'exercice :



# 6 Exercice 5

## 6.1 Objectif de l'exercice

Dans cet exercice 5, on va principalement travailler sur les variables. On va devoir en concaténer, en additionner etc.

## 6.2 Explications / Description

Pour déclarer une variable dans javascript, il existe 4 manières. La première est celle-ci :

```
maVar;
```

Les fonctions sont identiques à celle-ci :

```
var maVar;
```

Les deux ne sont plus utilisés étant donné que l'on peut déclarer deux fois la même variable et on a donc trop de libertés.

Maintenant, on utilise le mot-clé let :

```
let maVar;
```

C'est celle qui est la plus utilisée étant donné que comparé aux deux autres, elle ne peut pas être déclarée une deuxième fois.

Et pour finir il y a le mot-clé `const` :

```
const maVar="Valeur";
```

C'est une sorte de constante étant donné que l'on ne peut pas réaffecter la variable mais on peut modifier par exemple si c'est un tableau on pourra modifier les cellules.

Pour la convention de nommage, comme en java, les variables basiques s'écriront en dos de chameau et les constantes en majuscules.

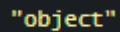
Pour le premier type de déclaration, c'est celle que l'on va utiliser pour déclarer des variables globales.

Dans javascript, il existe que 4 types de base : « number » pour un nombre, « string » pour une chaîne de caractères, « boolean » pour un booléen et « object » pour n'importe quel objet.

Pour connaître le type d'une variable, on peut utiliser `typeof`. Voici un exemple :

```
console.log(typeof(maVar));
```

Et voici le résultat dans la console pour un objet :



## 6.3 Extrait de code

Pour l'extrait de code, voici le code en entier commenté par partie :

```
// Vider la console
console.clear();

// Assigner et afficher la variable a
let a;
console.log(a);
a = 15;
console.log("Ma variable a = " + a);

// Identique pour la b
let b = 9;
console.log("Ma variable b = " + b);

// Calcul entre les deux variables
console.log(`${a} + ${b} = ${a + b}`);
console.log(`${a} - ${b} = ${a - b}`);
console.log(`${a} * ${b} = ${a * b}`);
console.log(`${a} / ${b} = ${a / b}`);

// Concaténation de String
a = "Bonjour";
b = " les amis";
console.log(a + b);
console.log(`${a}${b}`);

// Vérifications entre des booléen
a = true;
b = false;
```

```

console.log(`${a} AND ${b} = ${a && b}`);
console.log(`${a} OR ${b} = ${a || b}`);

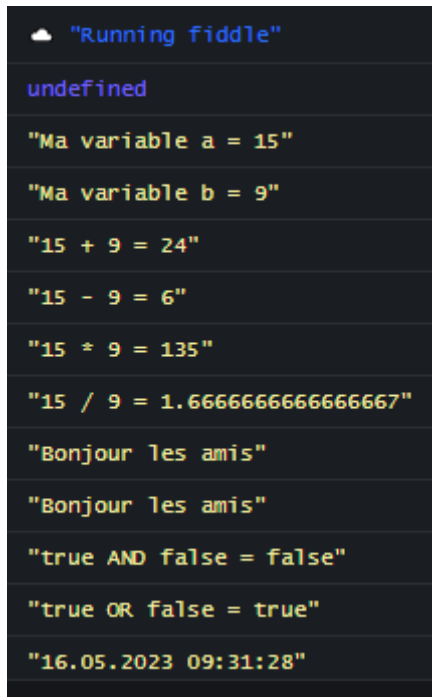
// Calcul avec les Dates
a = new Date(Date.now());
b = new Date();
b.setDate(a.getDate() - 61);
console.log(a.toLocaleDateString() + " " + a.toLocaleTimeString());
console.log(b.toLocaleDateString() + " " + a.toLocaleTimeString());
console.log(a.toLocaleDateString());
console.log(a.toLocaleTimeString());
console.log(b.toLocaleDateString());
console.log(b.toLocaleTimeString());

// Observation des différents types
a = Math.PI;
b = "bonjour";
let c = true;
let d = new Date(Date.now());
let e;
console.log(typeof(a));
console.log(typeof(b));
console.log(typeof(c));
console.log(typeof(d));
console.log(typeof(e));

```

## 6.4 Capture

Et dans les captures, voici une capture avec une partie des résultats :



```

Running fiddle
undefined
"Ma variable a = 15"
"Ma variable b = 9"
"15 + 9 = 24"
"15 - 9 = 6"
"15 * 9 = 135"
"15 / 9 = 1.6666666666666667"
"Bonjour les amis"
"Bonjour les amis"
"true AND false = false"
"true OR false = true"
"16.05.2023 09:31:28"

```

Et maintenant les résultats complet :

```
Undefined
"Ma variable a = 15"
"Ma variable b = 9"
"15 + 9 = 24"
"15 - 9 = 6"
"15 * 9 = 135"
"15 / 9 = 1.6666666666666667"
"Bonjour les amis"
"Bonjour les amis"
"true AND false = false"
"true OR false = true"
"16.05.2023 09:31:28"
"16.03.2023 09:31:28"
"16.05.2023"
"09:31:28"
"16.03.2023"
"09:31:28"
"number"
"string"
"boolean"
"object"
"undefined"
```

## 7 Exercice 6

### 7.1 Objectif de l'exercice

Dans l'exercice, nous allons utiliser un if ou un switch et on va aussi de nouveau utiliser les dates.

### 7.2 Explications / Description

Dans cet exercice, on devra afficher le jour de la semaine lorsque l'on appuie sur le bouton. Donc d'abord il faudra récupérer le numéro du jour actuel. Pour cela on va initialiser une date et on va récupérer le jour de la semaine grâce à `getDay()`.

Pour insérer du texte dans une balise html existante, on va utiliser la balise `document.getElementById` avec l'ID de l'élément. Et ensuite on va utiliser le `innerHTML` pour écrire dans l'élément ce que l'on va mettre après le `innerHTML`. Voici un exemple :

```
document.getElementById("info").innerHTML = s + "mardi";
```

Pour ajouter du code c'est de la même façon. Maintenant pour créer un tableau, on va créer une variable et on va mettre les crochets. Dans les crochets on pourra mettre les valeurs que l'on veut. Voici un exemple de création :

```
const tab = ["dimanche", "lundi",
"mardi", "mercredi", "jeudi", "vendredi", "samedi"];
```

Ensuite on va pouvoir effectuer plein d'action différentes en utilisant le nom du tableau un point et le nom de l'action. Pour la première action, c'est pour ajouter un élément au début d'un tableau. Pour cela on va utiliser le unshift. Pour ajouter à la fin, on va utiliser le push. Pour supprimer le premier, on utilise le shift et pour supprimer le dernier le pop. Voici les commandes :

```
tab.push('Eric'); // ajoute Eric à la fin du tableau
tab.unshift('AA'); // ajoute AA au début du tableau
tab.pop(); // efface le dernier élément du tableau
tab.shift(); // efface le premier élément du tableau
```

Pour récupérer la valeur d'un case précise d'un tableau, on va mettre le tableau et entre crochets le numéro de la case. Voici un exemple :

```
tab[2]
```

Et pour finir pour afficher toutes les valeurs d'un tableau, on va faire une boucle foreach comme la suivante :

```
noms.forEach(nom => console.log (nom));
```

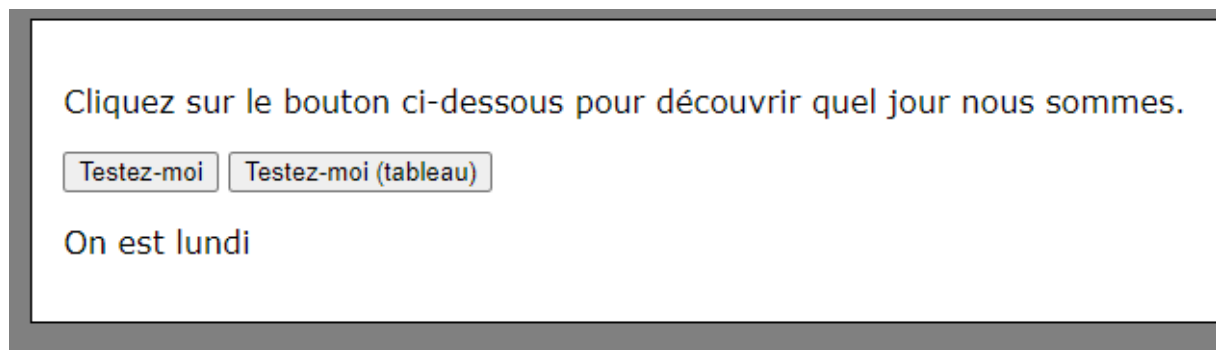
## 7.3 Extrait de code

Comme extrait de code, je vais mettre la façon la plus simple pour afficher le jour de la semaine. Dans cette façon on va utiliser un tableau et on va récupérer la valeur du tableau qui correspond à la position du jour correspondant :

```
function afficherJourSemaineTableau() {
  let date = new Date();
  let s = "On est ";
  const tab = ["dimanche", "lundi",
    "mardi", "mercredi", "jeudi", "vendredi", "samedi"];
  const jour = tab[date.getDay()]
  document.getElementById("info").innerHTML = s + jour
}
```

## 7.4 Capture

Voici la capture du site avec un bouton pour chaque façon de faire :



## 8 Exercice 7

### 8.1 Objectif de l'exercice

Dans cet exercice, on va principalement utiliser les boucles. On aura donc la boucle for, la boucle while et la boucle do-while.

### 8.2 Explications / Description

Pour toutes les boucles, c'est comme dans java. Mais voici quand même pour la boucle for (on connaît le nombre d'itération) :

```
for (let i = 0; i < 5; i++) {  
  s += "  i = " + i + "<br/>";  
}
```

Maintenant pour la boucle while (exécuter tant que...) :

```
let i = 0;  
while (i<5) {  
  s += "  i = " + i + "<br/>";  
  i++;  
}
```

Et pour finir la boucle do-while (exécuter tant que... mais au moins une fois) :

```
do {  
  s += "  i = " + i + "<br/>";  
  i++;  
} while (i<5);
```

### 8.3 Extrait de code

Pour cet exercice, il n'y aura pas d'extrait de code étant donné que le code est déjà dans le point précédent pour l'explication.

### 8.4 Capture

Et pour les capture d'écrans, voici d'abord la page de base :

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

Et voici l'exemple lorsque l'on appuie sur un bouton :

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
```

--> A utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)

## 9 Exercice 8

### 9.1 Objectif de l'exercice

Dans cet exercice, nous allons découvrir et apprendre à utiliser les tableaux de JSON dans un js.

### 9.2 Explications / Description

Pour parcourir un tableau json, on va utiliser une boucle. On peut utiliser différents types de boucle et voici un exemple où l'on va faire une itération pour chaque champ dans le tableau personne :

```
for (const f in personne) {  
    txt += personne[f];  
}
```

Pour récupérer la valeur d'un champ, on va faire le nom du tableau avec entre crochets le champ étant donné que le f est le numéro du champ dans le tableau de json.

### 9.3 Extrait de code

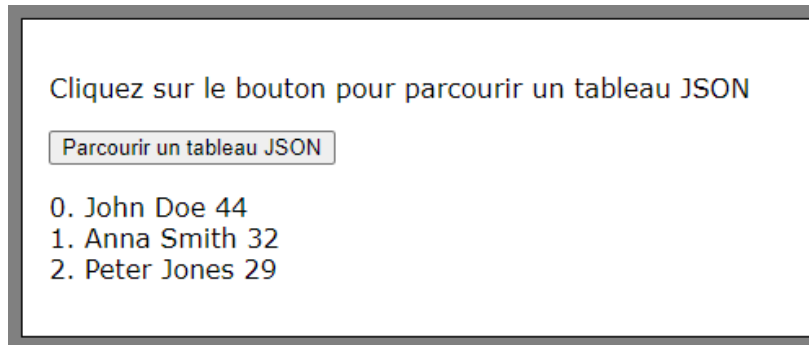
Voici un exemple de code avec deux boucles imbriquées étant donné que l'on a des tableaux dans un tableau :

```
for (const pers in json.personnes) {  
    txt += pers + ". ";  
    for (const f in json.personnes[pers]) {  
        txt += json.personnes[pers][f] + " ";  
    }  
    txt += "<br/>"  
}
```



## 9.4 Capture

Et voici le résultat au niveau graphique que l'on obtient :



## 10 Exercice 9

### 10.1 Objectif de l'exercice

Dans cet exercice, nous allons voir comment créer des objets dans notre js.

### 10.2 Explications / Description

Dans cet exercice, nous allons tester 4 différentes méthodes mais la meilleure est la dernière étant donné que c'est avec une classe. Avec une classe, c'est presque comme dans java. Voici un exemple de création de classe :

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

Ensuite si on veut ajouter une méthode à la classe, on va juste déclarer la méthode dans la classe comme ceci :

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

Pour créer un objet à partir de cette classe, on va faire un new et le nom de la classe :

```
let p1 = new Eleve("Julien", "Tartampion", 18);
```

### 10.3 Extrait de code

Comme extrait de code, voici la création d'une classe complète :

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

```

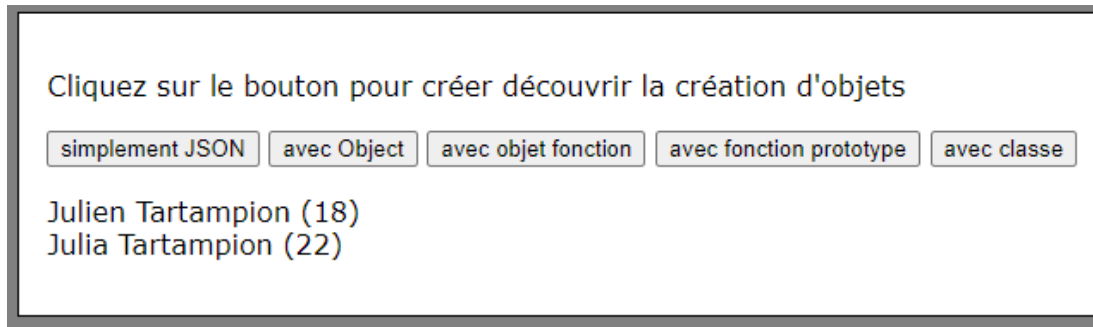
    this.age = age;
  }

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

```

## 10.4 Capture

Et pour les captures, voici lorsque l'on utilise la méthode avec la classe :



## 11 Exercice 10

### 11.1 Objectif de l'exercice

Dans cet exercice, nous allons commencer à utiliser la programmation orienté objet. L'objectif sera de pouvoir ajouter ou supprimer des personnes avec des boutons mais aussi de reprendre les informations d'une personne.

### 11.2 Explications / Description

Au début de notre js qui fait office de contrôleur (indexCtrl.js), on va utiliser une commande qui va nous permettre de faire des actions dès que le document a fini de charger. Pour cela on va utiliser le document.onreadystatechange. Voici un exemple :

```

document.onreadystatechange = function () {
  if (document.readyState === "complete") {
    _afficherPersonnes();
  }
};

```

On utilise les 3 égal comme ça on va contrôler que les deux éléments ont la même valeur mais aussi du même type.

Toujours dans notre indexCtrl.js, on va créer des méthodes pour la vue donc pour afficher les informations d'une personne, d'en ajouter ou d'en supprimer une. Elles utilisent juste une méthode du worker.js qui va vraiment faire la logique métier.

On aura aussi un bean Personne. Et dans le worker on aura donc trois méthodes qui vont faire des actions sur un tableau de personne.

## 11.3 Extrait de code

Voici les trois méthodes du worker qui vont contenir la logique métier :

```
function _trouverPersonne(p) {
  let idx = -1;
  for (let i = 0; i < personnes.length; i++) {
    if (personnes[i].toString() == p.toString()) {
      idx = i;
    }
  }
  return idx;
}

function ajouterPersonne(p) {
  let idx = _trouverPersonne(p);
  if (idx == -1) {

    personnes.push(p);
  }
  personnes.sort();
  _afficherPersonnes();
}

function supprimerPersonne(p) {
  let idx = _trouverPersonne(p);
  if (idx >= 0) {
    personnes.splice(idx, 1);
  }
  console.log(idx);
  _afficherPersonnes();
}
```

Voici aussi les deux méthodes du contrôleur qui vont afficher les informations d'une personne et lire les informations d'une personne :

```
function _afficherInfosPersonne(p) {
  let prenom = p.prenom;
  let nom = p.nom;
  let age = p.age;
  document.getElementById("prenom").value = prenom;
  document.getElementById("nom").value = nom;
  document.getElementById("age").value = age;
}

function _lireInfosPersonne() {
  let p = null;
  let prenom = document.getElementById("prenom").value ;
  let nom = document.getElementById("nom").value ;
  let age = document.getElementById("age").value ;
  p = new Personne(prenom, nom, age) ;
  return p;
}
```

## 11.4 Capture

Voici la capture lorsque l'on ajoute une nouvelle personne :

— Veuillez introduire une nouvelle personne ou sélectionner une existante : —

Prénom:

Nom:

Âge:

- [Anna Smith \(32\)](#)
- [John Doe \(44\)](#)
- [Peter Jones \(29\)](#)
- [Rémi Carrard \(16\)](#)

Et maintenant si on reprend les données d'une personne déjà existante :

— Veuillez introduire une nouvelle personne ou sélectionner une existante : —

Prénom:

Nom:

Âge:

- [Anna Smith \(32\)](#)
- [John Doe \(44\)](#)
- [Peter Jones \(29\)](#)
- [Rémi Carrard \(16\)](#)

## 12 Exercice 11

### 12.1 Objectif de l'exercice

L'exercice 11 est une copie de l'exercice 10 mais on devra le faire avec des classe et pas des prototypes.

### 12.2 Explications / Description

Pour créer et instancier une classe, on a déjà vu dans l'exercice 9. Pour créer notre objet contrôler, on va utiliser le document.onreadystatechange. On devra donc instancier notre contrôleur mais on ne va pas mettre de mot clé devant le nom de la variable lorsqu'on la déclare pour que la variable soit globale.

Et pour le reste du document, on va mettre les méthodes dans une classe pour chaque fichier. Cependant le contenu des méthodes ne va pas changer.

## 12.3 Extrait de code

Comme extrait de code, voici tout d'abord le document.onreadystatechange :

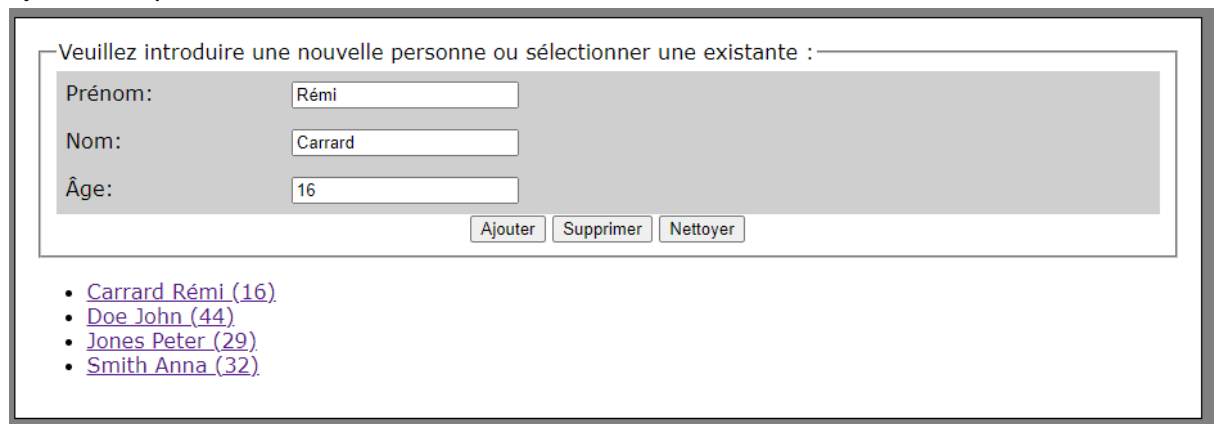
```
document.onreadystatechange = function () {  
    if (document.readyState === "complete") {  
        window.ctrl = new Ctrl(); // ou ctrl = new Ctrl();  
    }  
};
```

Voici de nouveau le bean personne mais avec une classe :

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

## 12.4 Capture

L'interface sera donc identique à l'exercice précédent mais voici quand même lorsque l'on ajoute une personne :



Veuillez introduire une nouvelle personne ou sélectionner une existante :

Prénom:

Nom:

Âge:

- [Carrard Rémi \(16\)](#)
- [Doe John \(44\)](#)
- [Jones Peter \(29\)](#)
- [Smith Anna \(32\)](#)

## 13 Exercice 12

### 13.1 Objectif de l'exercice

Pour cet exercice, on va découvrir les différentes déclarations de fonction possibles avec javascript.

### 13.2 Explications / Description

Il existe donc trois types de déclaration de fonctions. Le tout premier est la déclaration basique d'une fonction. Pour cela on va juste mettre function et le nom de la fonction. Voici un exemple :

```
function a("paramètre") {  
    let val = 1;  
    console.log(val) ;  
}
```

Ensuite on aura une fonction semblable à celle-ci mais la différence est qu'elle sera en anonyme. Pour faire cette fonction, on va mettre en quelque sorte la fonction dans une variable mais on ne va pas donner de nom à la fonction. Voici un exemple :

```
let b = function("paramètre") {  
    let val = 2;  
    console.log(val) ;  
} ;
```

Et pour finir la dernière fonction est la fonction flèche. Son avantage est qu'elle est rapide à implémenter. Dans cette fonction, on va d'abord mettre les paramètres, après une flèche => et pour finir les actions que l'on va faire dans la fonction. Voici un exemple de déclaration où l'on fait qu'une seule action :

```
let c = (a,b) => a + b ;
```

Et maintenant une où l'on fait plusieurs actions :

```
let d = (a,b) => {  
    let somme = a+b;  
    return somme;  
}
```

Pour appeler chacune de ces trois fonctions, on va mettre le nom de la fonction et les paramètres. Voici un exemple d'appel :

```
console.log(c(3,4)) ;
```

Il est aussi possible de créer des fonctions IIFE qui sont des fonctions qui vont être exécutées directement. Voici un exemple de fonction IIFE sur la base d'une fonction basique :

```
(function() {  
    let val = 3;  
    console.log(val) ;  
})();
```

Et maintenant voici la même chose mais sur la base d'une fonction en flèche :

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

### 13.3 Extrait de code

Pour cet exercice, il n'y aura pas de partie code étant donné que l'on découvre les fonctions et que l'on n'a pas réellement de programme à réaliser.

### 13.4 Capture

Pour les captures, c'est pareil il n'y en aura pas.

## 14 Exercice 13

### 14.1 Objectif de l'exercice

Dans l'exercice 13, nous allons découvrir et utiliser les cookies.

### 14.2 Explications / Description

Un cookie est une information qui est stocké sur la station cliente. L'information sera gardée entre la navigation des pages et cela peut être utile par exemple pour des identifiants même si c'est assez dangereux. Certains site web utilisent les cookies pour stocker les préférences d'un clients. C'est pourquoi il est possible de les refuser sur la majorité des site web.

Pour créer un cookie, on va utiliser le `document.cookie` et on va pouvoir lui assigner une valeur comme par exemple une chaîne de caractères. Voici un exemple :

```
document.cookie = "cookie_exercice_13=" + contenu + "; expires=" +  
date.toUTCString() + "; path=/;";
```

Et pour récupérer les éléments de notre cookie, on va aussi utiliser le `document.cookie` mais cette fois on va rajouter un `split` pour pouvoir récupérer uniquement la valeur souhaitée. Voici par rapport à l'exemple ci-dessus pour récupérer le contenu :

```
let listString = document.cookie.split("=");  
let contenu = listString[1].split(" ");
```

Dans cet exercice, on va aussi utiliser des tableaux en JSON. Donc pour convertir une chaîne de caractère en JSON, on va tout simplement créer notre JSON et on va mettre nos clés et dans la valeur on va pouvoir mettre notre chaîne de caractère. Par contre pour récupérer des chaînes de caractères dans un JSON, on va faire le nom du tableau un point et le nom de la clé. Voici un exemple où l'on va récupérer le prénom et le nom dans un JSON :

```
document.getElementById("prenom").value = newJson.prenom;  
document.getElementById("nom").value = newJson.nom;
```

## 14.3 Extrait de code

Pour l'extrait de code de cet exercice, il y aura la fonction pour ajouter un cookie et la fonction pour récupérer un cookie. Voici donc d'abord celle pour ajouter :

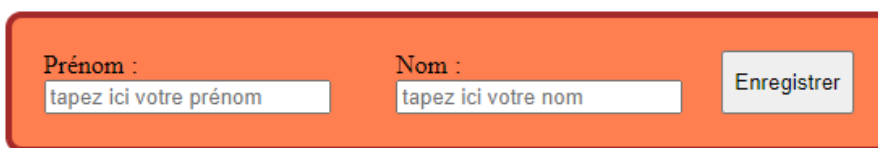
```
function setCookie(contenu, minutes) {  
    let date = new Date();  
    date.setTime(date.getTime() + 60000*minutes);  
    console.log(date.toUTCString());  
    document.cookie = "cookie_exercice_13=" + contenu + "; expires=" +  
date.toUTCString() + "; path=/;";  
    console.log("Data saved in cookie");  
}
```

Et voici maintenant la fonction pour aller récupérer le cookie :

```
function getCookie() {  
    if (document.cookie.length > 0) {  
        let listString = document.cookie.split("=");  
        let contenu = listString[1].split(" ");  
        let newJson = {  
            prenom: contenu[0],  
            nom: contenu[1]  
        }  
        document.getElementById("prenom").value = newJson.prenom;  
        document.getElementById("nom").value = newJson.nom;  
        console.log("Data retrieved from cookie");  
    }  
}
```

## 14.4 Capture

Et pour les captures d'écran, voici ce que l'on a lorsque l'on n'a pas encore enregistré de cookie :



Prénom :  Nom :

Et voici ce que l'on obtient lorsque l'on a enregistré un cookie et que l'on arrive sur la page :



Prénom :  Nom :



# 15 Exercice 14

## 15.1 Objectif de l'exercice

Dans cet exercice, on ne va pas vraiment faire du code étant donné que l'on aura un site internet avec des slides pour expliquer les principes de base de jquery

## 15.2 Explications / Description

Jquery est une librairie disponible pour javascript. Elle permet de faciliter énormément les échanges entre du code javascript et du code HTML. Elle simplifie énormément la manipulation du DOM.

Dans jquery, on va tout le temps utiliser le \$ avec des parenthèses. Cela nous permet de sélectionner un élément dans notre HTML. C'est donc grâce à cela qu'on peut reconnaître du code jquery.

Les deux tâches principales d'un code jquery est de retrouver un ou plusieurs éléments dans un fichier HTML et ensuite de faire une action avec ce ou ces éléments.

Pour retrouver un élément de notre HTML, on va mettre le nom de l'élément entre parenthèses dans le \$(""). Voici un exemple où l'on va récupérer l'ensemble des div du document :

```
$("#div")
```

Ensuite si on veut sélectionner un ID, on va mettre un # juste avant le nom. Et pour une classe CSS, on va rajouter un point avant.

Lorsque l'on lance la page, la fonction ready va être lancée. Voici la structure que l'on aura.

```
$(document).ready(function() {  
    // Code jQuery ici  
});
```

Dans jquery, on pourra sélectionner des éléments par rapport à un autre. Par exemple on peut sélectionner l'élément parent d'un élément. Il existe 5 sélections comme celle-ci. On aura donc parent() qui sélectionne le parent, next() qui sélectionne le prochain élément, prev() pour celui d'avant, children() pour les éléments enfants et siblings() pour les éléments frères. Voici un exemple d'utilisation de ces sélections :

```
$("#button").parent().css("border", "3px solid red");
```

Ensuite si on veut modifier du css pour un élément, on va mettre un point et css. Ensuite on va mettre entre parenthèses des crochets et ensuite on pourra mettre du code css comme si on était dans le fichier. Voici un exemple :

```
$("#li a").css({  
    color: "red",  
    fontWeight: "bold"  
});
```

Pour ajouter un élément après celui sélectionné, on va utiliser le `after`. Et si on veut mettre un élément avant ce sera le `before` :

```
$("#a[target=_blank]")
  .after("<img src='images/open.png' />");
```

Sinon on aura aussi le `append` pour ajouter directement du texte dans la balise de l'élément. Voici un exemple :

```
$("#a[target=_blank]")
  .append("(mon texte)");
```

Ensuite si on veut supprimer un élément, on va utiliser le `remove`. Pour cela ce sera comme pour ajouter mais on ne va rien mettre entre parenthèses. Voici l'exemple :

```
$("#a[target=_blank"]').remove();
```

Pour les formulaires, on peut utiliser l'évènement `submit()`. Sinon on aura aussi l'évènement `click()` qui va nous permettre de faire une action dès que l'on va cliquer sur l'élément sélectionné. Et dans le même style on a `hover` mais ici l'action sera faite uniquement lorsque l'on aura la souris sur l'élément.

L'évènement `toggle()` permet d'activer ou de désactiver certains éléments. Par exemple dans l'exemple suivant, on va activer ou désactiver les éléments lorsque l'on va cliquer sur le menu :

```
$("#a.menu").click(function(){
  $(this).next().toggle();
  return false;
});
```

On peut aussi utiliser le `animate` pour faire une petite animation durant l'action :

```
$("#li").hover(function(){
  $(this).animate({marginLeft: 38, marginRight: 0});
}, function(){
  $(this).animate({marginLeft: 18, marginRight: 18});
});
```

On va utiliser aussi l'élément `live()` qui va remplacer le `hover`. Cela nous permet de faire ces actions aussi sur les nouveaux éléments et pas seulement sur ceux de base du fichier. Voici un exemple :

```
$("#a.menu").live("hover", function(){
  $(this).next().toggle(200);
  return false;
});
```

Ensuite pour cacher et afficher un élément, on va utiliser `hide()` et `show()`. Le premier argument est la vitesse de l'action et le deuxième est ce que l'on va faire lorsque l'on aura fini l'action. Dans l'exemple suivant on va d'abord cacher l'élément et ensuite on va le réafficher :

```
$("#div.block").hide("slow", function(){
  $(this).show("slow");
});
```

Pour lire un fichier avec ajax, on va faire `$.ajax({})`. Entre les accolades, on va mettre tout d'abord l'url du fichier XML et ensuite on va faire une fonction. Voici la déclaration avec un exemple de fonction :

```
$.ajax({
  url: "file.xml",
  success: function( xml ) {
    $(xml).find("tab").each(function() {
      $("ul").append(
        "<li>" + $(this).text() + "</li>");
    });
  }
});
```

Ensuite pour un fichier JSON, on va utiliser la méthode `$.getJSON()`. Dans cette méthode, on va d'abord mettre en paramètre le fichier json et la fonction que l'on va faire. Dans la fonction on peut par exemple lire tout les éléments dans le json et les ajouter à une liste. Voici l'exemple en code :

```
$.getJSON("file.json", function( obj ) {
  for ( var prop in obj ) {
    $("ul").append(
      "<li>" + prop + ": " + obj[prop] + "</li>");
  }
});
```

Pour décider de charger un fichier HTML autre part, on va utiliser le `load()`. On va donc d'abord mettre l'élément dans lequel stocker et ensuite le fichier HTML. Voici un exemple :

```
$("#div.load").load("file.html");
```

Et pour finir pour intégrer jquery dans notre projet, on va mettre dans le head du HTML (comme pour le js) cette ligne :

```
<script src='http://code.jquery.com/jquery.js'></script>
```

## 15.3 Extrait de code

Dans cet exercice, il n'y aura pas d'extrait de code étant donné que l'on n'a pas fait de code nous-même.

## 15.4 Capture

Et pour les captures, c'est pareil il n'y en aura pas pour cet exercice.

## 16 Exercice 15

### 16.1 Objectif de l'exercice

Dans l'exercice 15, on va commencer à utiliser jquery dans un vrai programme. Le principe est de changer la couleur de fond de la page avec jquery et avec une animation.

### 16.2 Explications / Description

Déjà au tout début il faudra récupérer la valeur de la couleur que l'on a choisi. Pour cela on va récupérer notre liste avec l'id couleurs et ensuite on va obtenir sa valeur avec « .val() ».

```
$("#couleurs").val()
```

Ensuite pour changer la couleur de fond, on va utiliser le « .css » et on va modifier l'attribut background-color. On va utiliser aussi color pour changer la couleur de l'écriture. Voici un exemple du css :

```
$(#container).css({
    "background-color": "green",
    "color": "white"
})
```

Et pour finir pour mettre une animation de disparition et de réapparition, on va utiliser le fadeOut durant une seconde pour qu'il disparaisse, après on fait le css et pour finir on fait un slideToggle pour qu'il réapparaisse en glissant depuis le haut. Voici un exemple de code où l'on utilise le fadeOut et le slideToggle :

```
$("#container").fadeOut(1000, function () {
    $(this).css({
        "background-color": couleur,
        "color": "black"
    }).slideToggle(1000);
});
```

### 16.3 Extrait de code

Pour cet exercice, le seul élément intéressant est la fonction changerCouleur de la classe Ctrl. Donc voici tout le code de cette fonction :

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

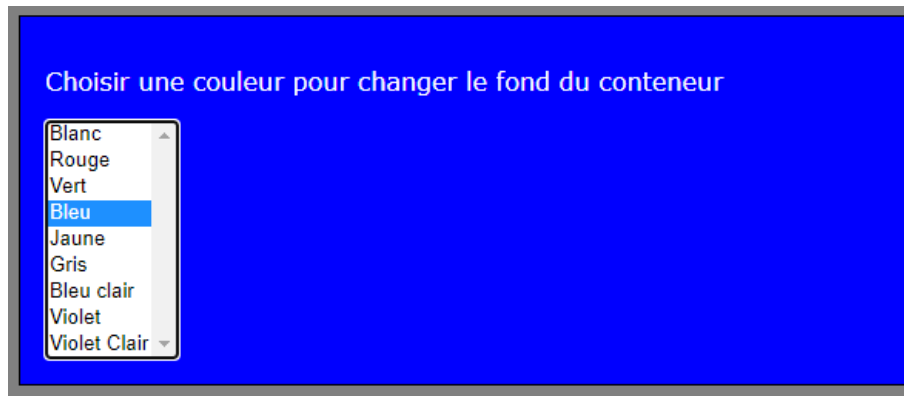
```

        "color": "black"
    }) .slideToggle(1000);
  });
}
}

```

## 16.4 Capture

Et pour les captures d'écran, voici juste un exemple de l'interface étant donné que pour montrer les animations, ce n'est pas très pratique :



## 17 Exercice 16

### 17.1 Objectif de l'exercice

Dans cet exercice, nous allons utiliser plein de fonctions différentes dans jQuery. Cela nous permettra de nous familiariser encore avec cela.

### 17.2 Explications / Description

Pour placer un écouteur sur un élément, on va déjà sélectionner l'élément dans jQuery et ensuite on va utiliser le "on" et on va mettre ensuite click pour spécifier que c'est lorsque l'on va cliquer. Et pour finir on va mettre la fonction que l'on va faire lorsque l'élément sera cliqué. Voici un exemple d'écouteur :

```

$("#btnShow").on("click", function () {
    ctrl.afficher();
});

```

Comme déjà dit plus tôt pour sélectionner un élément de notre fichier, on va mettre un \$ et le nom de l'id, de la classe ou de l'élément. Par exemple dans l'exemple ci-dessus, on va sélectionner l'élément avec l'id btnShow.

Pour ajouter un style css à un élément sélectionné, on peut soit directement ajouter avec le .css ou bien on ajoute une classe qui a déjà un style avec le .addClass. Avec cette dernière méthode, c'est possible de supprimer le style css en enlever juste la classe de l'élément avec .removeClass. il est aussi possible de mettre le .toggleClass qui va permettre d'activer et de désactiver une classe sur un seul bouton. Voici maintenant un exemple d'ajout de classe :

```

$("div>ol>li").first().addClass("boldBlueText");

```

Et maintenant un exemple de suppression :

```
$("div>ol>li").first().removeClass("boldBlueText");
```

Pour récupérer des valeurs d'un formulaire dans jQuery, on va utiliser le `.val()` après avoir sélectionné l'élément du formulaire. Voici un exemple :

```
$("#textToInsert").val()
```

Et on peut aussi contrôler si une checkbox est cochée avec le `.is(":checked")`. Voici un exemple :

```
if ($("#textBold").is(":checked")) {}
```

Pour ajouter des éléments à notre DOM, on va pouvoir utiliser tout simplement le `append`. Voici un exemple :

```
$("#textes").append("<p>" + $("#textToInsert").val() + "</p>");
```

Et pour récupérer la valeur d'un attribut, on va utiliser le `.attr()` avec le nom de l'attribut. Voici un exemple :

```
ctrl.afficherPK($(this).attr("pk"));
```

Dans cet exercice, on a aussi utilisé plein d'animations différentes pour des éléments comme le `fadeIn/fadeOut` ou encore le `slideUp/slideDown` et pour finir le `show/hide/toogle`.

On a aussi utilisé des sélecteurs spéciaux comme `first()` pour le premier élément, `last()` pour le dernier, `even()` pour les éléments pairs ou `odd()` pour les éléments impaires.

Et pour finir on va aussi utiliser la fonction `alert` qui permet de faire une popup. Voici un exemple de déclaration de la fonction :

```
alert("La clé primaire de la personne cliquée est " + pk);
```

## 17.3 Extrait de code

Comme extrait de code, il n'y en aura pas énormément étant donné que c'est juste le nom des fonctions utilisé qui changent et elles ont déjà été mentionnés plus haut. Mais voici quand même la fonction la plus complexe qui permet d'ajouter du texte dans le document et le mettre en gras si la case est cochée. Voici le code :

```
insertText() {
    $("#textes").append("<p>" + $("#textToInsert").val() + "</p>");
    if ($("#textBold").is(":checked")) {
        $("#textes>p").last().addClass("boldBlueText");
    }
}
```

## 17.4 Capture

Voici l'interface que l'on pourra avoir avec les différents boutons qui ont des fonctions différentes.

**Exercice 16 - Fonctions jquery**

Afficher Cacher Afficher / Cacher

Veit 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 laborum 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

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 exercitati 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 co 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 éléments impaires

1. **Minim do culpa duis dolor cillum enim consectetur pariatur irure anim ve**
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
5. **Lorem mollit nulla cillum ea dolor consequat enim ut occaecat proident.**

Test ☒ Text en gras Insérer le texte Test

Test

Test

José Pascal Henri Sophie

Et voici maintenant la popup que l'on a avec la fonction alert :

**carrandr.emf-informatique.ch indique**

La clé primaire de la personne cliquée est 41

OK

## 18 Exercice 17

### 18.1 Objectif de l'exercice

Dans cet exercice, on va faire des présentations pour découvrir les deux webservices et les possibilités.

### 18.2 Explications / Description

On a d'abord deux webservices possibles. Le premier est REST et le deuxième SOAP.

#### 18.2.1 REST

Pour REST, on aura une architecture client-serveur avec des identifiants pour les ressources (URI). On va pouvoir faire 5 types de requêtes que l'on va faire avec http. Elles sont expliquées dans les points suivants sauf pour le PATCH que l'on ne va pas utiliser. Le plus souvent, on va représenter les données avec du JSON mais c'est aussi possible avec du XML et du HTML.

REST est stateless ce qui veut dire qu'il ne prend pas en compte les requêtes précédentes. Il répond également avec du code.

##### 18.2.1.1 GET

Le GET est la fonction qui va nous permettre d'aller récupérer les données. Si on met des paramètres, ils seront affichés dans le lien directement après un ?.

### 18.2.1.2 POST

Le POST est la fonction pour ajouter un nouvel élément sur le serveur. On va donc donner les valeurs directement dans la requête (dans le body).

### 18.2.1.3 PUT

Le PUT va aussi envoyer des données au serveur mais cette fois c'est pour modifier un enregistrement. Les valeurs seront également dans la requête.

### 18.2.1.4 DELETE

Et pour finir le DELETE permet comme son nom le dit de supprimer des données. On va mettre les paramètres dans le lien pour spécifier quoi supprimer comme pour le GET.

## 18.2.2 SOAP

SOAP va utiliser l'encapsulation et va donc encapsuler les messages et le client va les décapsuler tout à la fin pour obtenir le résultat de la requête. Les données seront au format XML. Dans SOAP on aura également un élément qui s'appelle WSDL qui va spécifier ce que doit contenir le message. C'est une sorte de schéma XML et donc le XML devra correspondre au schéma.

# 19 Exercice 18

## 19.1 Objectif de l'exercice

Dans l'exercice 18, le but est de commencer à utiliser des vrais appels vers un serveur. Le serveur sera nous-même étant donné que l'on va appeler un PHP que l'on a dû créer.

## 19.2 Explications / Description

Dans cet exercice, on va donc faire des appels avec ajax. On aura eux possibilités d'appels. La première manière est avec un GET et la deuxième manière est avec un POST. La seule différence est que dans le PHP, cela sera différent si on utilise un GET ou un POST. Voici un appel ajax avec un GET :

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

Pour faire un appel ajax avec un POST, on change juste le GET avec un POST.

Les paramètres seront donc écrits dans la variable param qui est mise comme valeur dans data. Pour spécifier une méthode de retour lorsque c'est un succès, on va la mettre dans le success, et en cas d'erreur, dans le error.

Dans le fichier PHP, on devra ajouter une en-tête spéciale sinon on ne pourra pas faire de requête sur le PHP, il nous affichera une erreur avec le CORS. Voici ce que l'on doit ajouter tout en haut du fichier PHP :

```
header('Access-Control-Allow-Origin: *');
```



## 19.3 Extrait de code

Dans cet exercice, on aura plusieurs extraits de code. Voici tout d'abord le PHP qui va répondre à la requête :

```
<?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);
?>
```

Et voici maintenant la méthode qui va afficher la température en fahrenheit :

```
OKCelsius2Fahrenheit(data) {
    let temp = data.temperature;
    let affiche = temp !== "NaN" ? temp : "";
    // Afficher les degrés dans le formulaire
    $("#fahrenheit1").val(affiche);
}
```

La valeur de la variable temp va changer si on aura du XML et pas du JSON. Voici ce que l'on va mettre dans temp avec du XML :

```
$(data).find("temperature").text();
```

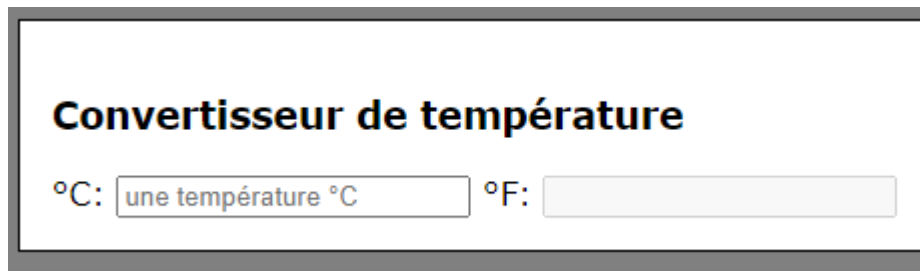
Et voici aussi le code que l'on va ajouter qui va nous permettre d'avoir plus d'informations en cas d'erreur :

```
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);  
}  
});  
}
```

## 19.4 Capture

Voici ce que l'on obtient lorsque l'on arrive sur l'application :



**Convertisseur de température**

°C:  °F:

Et voici maintenant lorsque l'on met une valeur :



**Convertisseur de température**

°C:  °F:

## 20 Exercice 20

### 20.1 Objectif de l'exercice

Dans l'exercice 20, nous allons créer un site web où la structure ressemble beaucoup à celle de notre projet. On aura donc de nouveau une vue séparée de la partie worker.

### 20.2 Explications / Description

Le principe de cette page est le SPA donc Single Page Application. On aura donc un seul fichier html où l'on va charger différentes vues sur celui-ci. On va souvent utiliser cela si on a un bout de la page qui reste identique sur toutes les pages et c'est seulement un bout de notre page qui change.

Pour charger notre code HTML d'une page vers notre page principale, on va utiliser la méthode `.load()` et on va mettre entre parenthèses le chemin vers la vue que l'on veut charger et aussi la méthode de callback. Voici un exemple où l'on appelle la vue :

```
$("#view").load("views/" + vue + ".html", function () {  
    // si une fonction de callback est spécifiée, on l'appelle ici  
    if (typeof callback !== "undefined") {  
        callback();  
    }  
});
```

De nouveau dans cet exercice on va utiliser le `$.ajaxSetup()`. Comme petit rappel, il nous permet de définir des valeurs différentes pour des erreurs. Cela va nous permettre de connaître la nature de l'erreur. Pour le code que l'on va mettre dedans les parenthèses, il y est à la fin du point code de l'exercice 18.

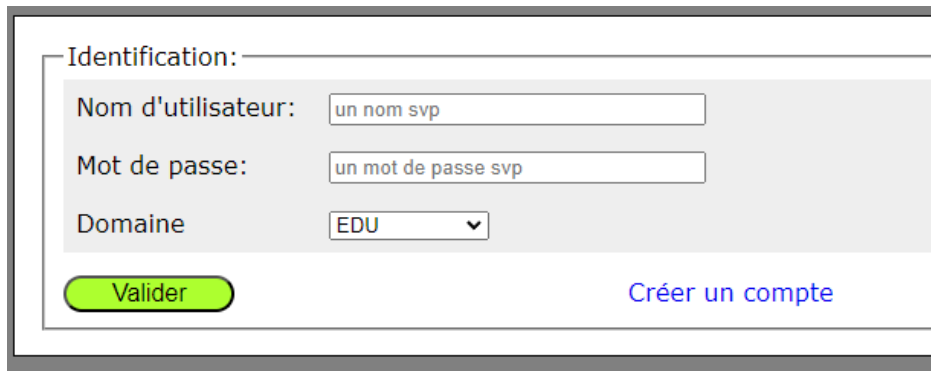
### 20.3 Extrait de code

Comme extrait de code, il n'y en a pas beaucoup étant donné que la plupart du code n'est pas compliqué et a déjà été expliqué dans les points précédents. Voici quand même un exemple de ce que l'on va faire dans le `indexCtrl` pour modifier la vue :

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

## 20.4 Capture

Et pour la partie des captures, il y a plein de fonctionnalités différentes. Voici d'abord ce que l'on a lorsque l'on arrive sur le site :



Identification: \_\_\_\_\_

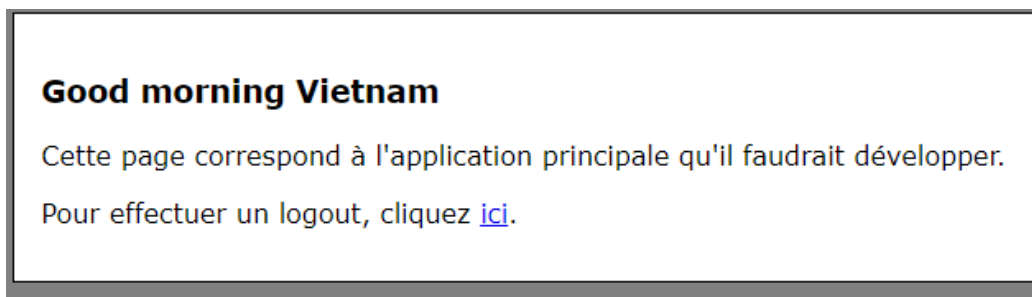
Nom d'utilisateur:

Mot de passe:

Domaine

[Créer un compte](#)

Voici maintenant si on se connecte avec des bons identifiants :

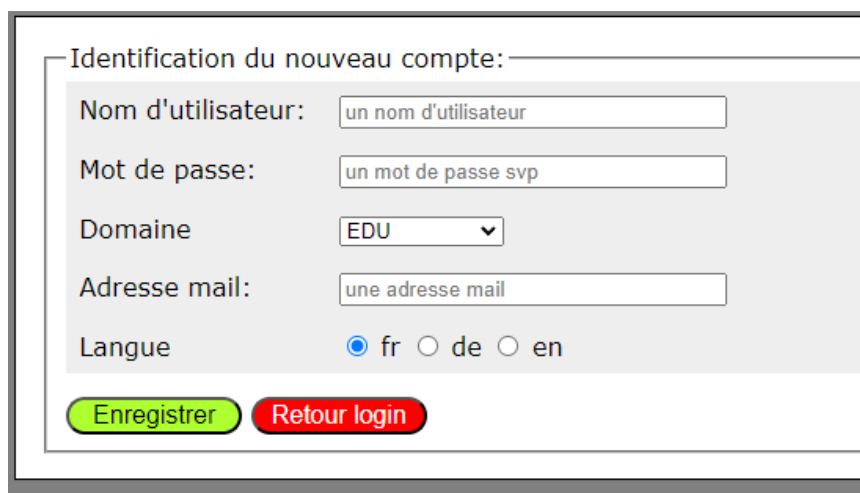


**Good morning Vietnam**

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

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

Maintenant lorsque l'on appuie sur créer un compte :



Identification du nouveau compte: \_\_\_\_\_

Nom d'utilisateur:

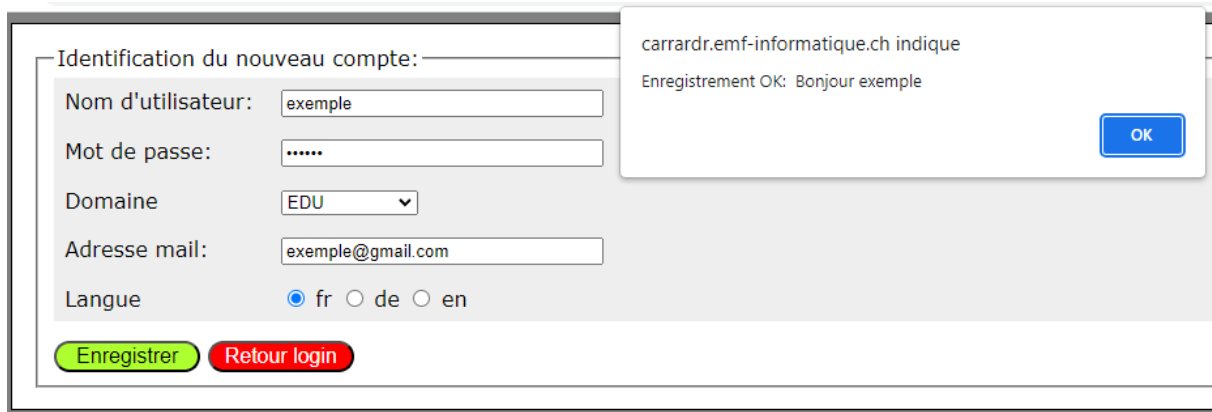
Mot de passe:

Domaine

Adresse mail:

Langue ☒ fr ☐ de ☐ en

Voici ce que l'on a lorsque l'on s'enregistre :



The image shows a web registration form titled "Identification du nouveau compte:". The form contains the following fields and options:

- Nom d'utilisateur:
- Mot de passe:
- Domaine:  (with a dropdown arrow)
- Adresse mail:
- Langue: ☒ fr ☐ de ☐ en

At the bottom of the form are two buttons: "Enregistrer" (green) and "Retour login" (red).

Overlaid on the top right of the form is a white message box with the text: "carradr.emf-informatique.ch indique Enregistrement OK: Bonjour exemple". It has a blue "OK" button in the bottom right corner.

Et si on appuie sur retour login au reviens à la page du début.

## 21 Projet

Cette avant-dernière partie du rapport concerne la grosse partie de ce module, le projet.

Ce projet sera un site web avec du javascript et qui va faire appel à une API externe. Il faudra donc récupérer les données sur l'API et les formater comme nous on le veut. On aura plusieurs fichier js avec l'utilisation du modèle MVC donc on aura des js en worker (ceux qui vont faire l'appel sur l'API) et des js en vue (ceux qui modifient la vue).

Pour l'API choisie, elle est sous ce lien : <https://api.openopus.org/>

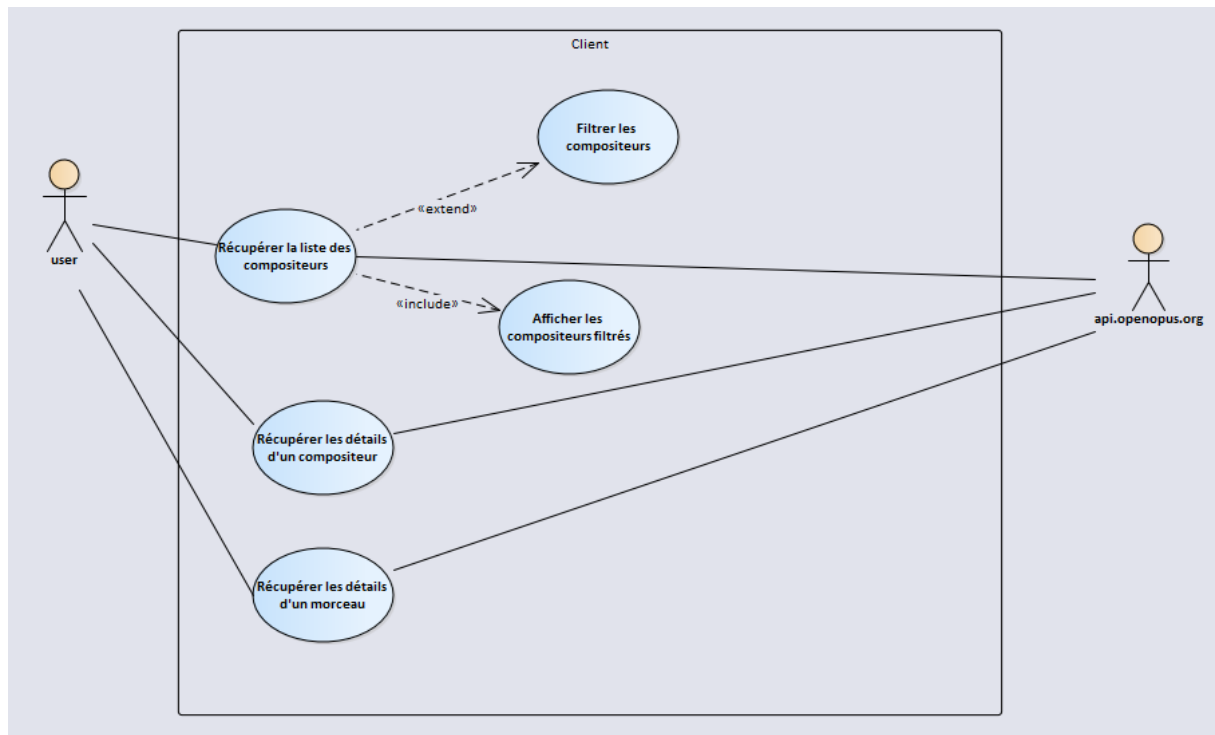
Mon site web permettra donc de répertorier une grande partie des compositeurs de différentes époques. On pourra afficher leurs détails mais aussi le détail des œuvres que chaque compositeur à écrite.

### 21.1 Analyse

L'analyse est la toute première partie de ce projet. C'est ici que l'on va commencer a réfléchir sur le fonctionnement de l'application et l'interface utilisateur du site.

### 21.1.1 Diagramme de use cases / Explications des cas

Pour débiter cette analyse, voici le use case de mon site qui nous permettra de savoir le fonctionnement du site et des possibilités que l'utilisateur a :



On pourra donc récupérer la liste des compositeurs et la filtrer pour l'afficher ensuite. Après on pourra récupérer les détail d'un compositeur. Et pour finir on peut récupérer les détails d'un morceau.

### 21.1.2 Maquette

Pour la maquette, voici la maquette pour la page avec la liste de tous les compositeurs :

On aura donc la liste des compositeurs à droite avec leur photo et à gauche un filtre pour les compositeurs.

Maquette de la page 'Liste des compositeurs' :

- En haut à gauche, un logo de note de musique.
- En haut à droite, le titre 'Liste des compositeurs :'.
- À gauche, une section 'Choisir un filtre'.
- À droite, une grille de 6 cartes (2 lignes de 3 cartes) :

Photo du compositeur	Photo du compositeur	Photo du compositeur
Nom du compositeur	Nom du compositeur	Nom du compositeur
Photo du compositeur	Photo du compositeur	Photo du compositeur
Nom du compositeur	Nom du compositeur	Nom du compositeur

Voici maintenant la page pour les détails du compositeur :

Maquette de la page 'Détails du compositeur' :

- En haut à gauche, un logo de note de musique.
- En haut à droite, le titre 'Nom du compositeur'.
- À gauche, une section 'Photo du compositeur' (contenant une image placeholder).
- À droite, des champs pour les informations :

Date de naissance:

Date de mort:

Epoque:

Liste des morceaux:

On aura quelques informations sur le compositeurs et aussi tous les morceau qu'il a écrit.

Et pour les détails d'un morceau, voici la page :



Nom du morceau

Genre:

Popularité:

Recommandé:

Compositeur:

On aura aussi quelques détails sur le morceau et le compositeur à la fin.

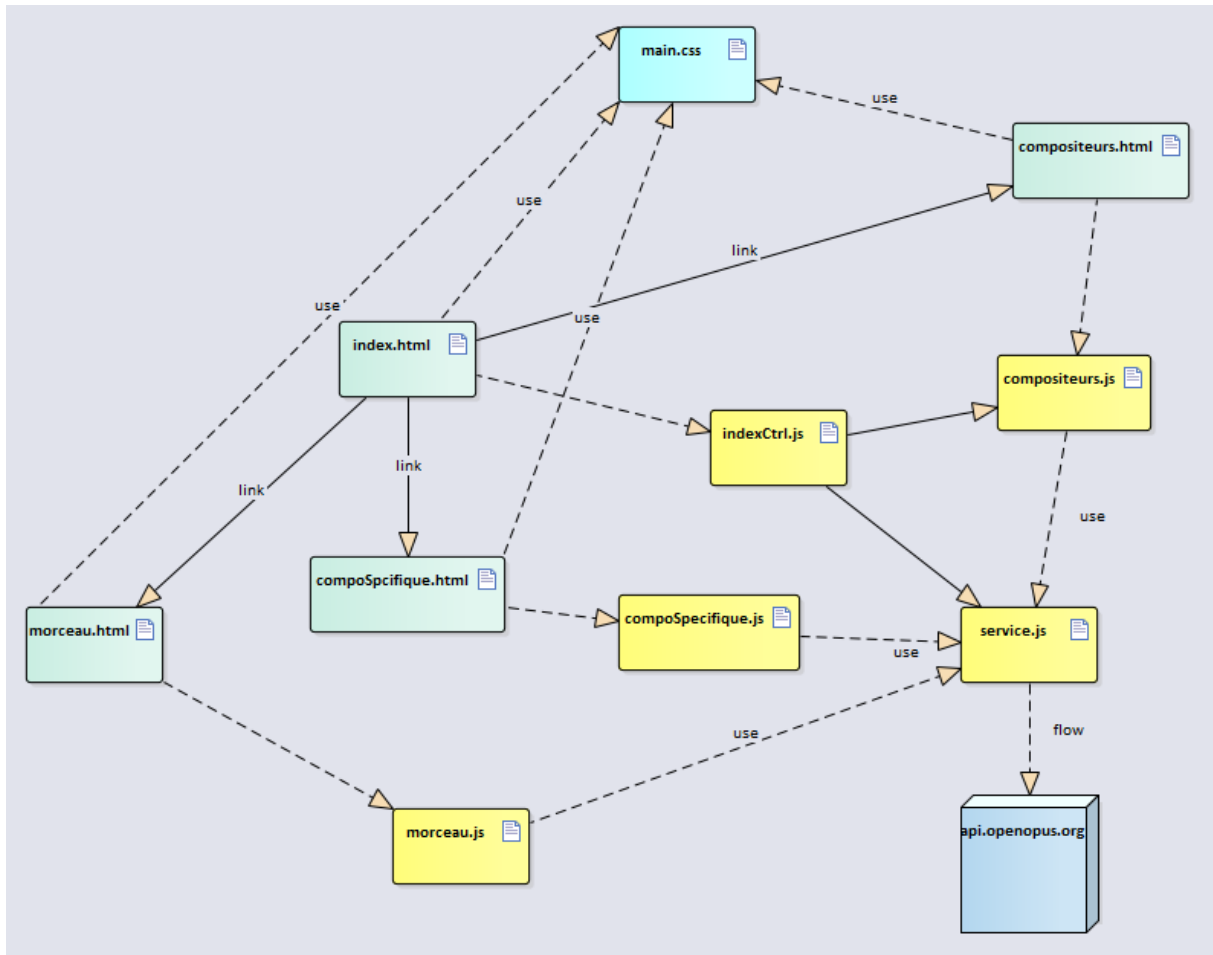


## 21.2 Conception

Pour la conception de ce projet, on aura uniquement un diagramme de navigation qui définit les liens entre les différents fichiers

### 21.2.1 Diagramme de navigation

Pour la conception, voici le diagramme de navigation :



Dans ce diagramme la partie la plus intéressante est la partie en bas à droite. On aura donc le `indexCtrl.js` qui va créer le `service.js` qui va lui communiquer avec le serveur qui contient l'API. Les trois autres fichiers JS utiliseront le `service.js`, et pour finir les fichiers HTML utiliseront le fichier JS correspondant.

## 21.3 Implémentation

Dans cette partie implémentation, on va avoir les bouts de code importants et un peu spéciaux. Tout le reste du code se trouve sur GitHub sous ce lien :

[Compositeur\\_project\\_api/code at main · CarrardR/Compositeur\\_project\\_api \(github.com\)](https://github.com/CarrardR/Compositeur_project_api)

### 21.3.1 HTML / CSS

Pour la partie HTML, il n'y a pas de code spécial. Mais pour la partie CSS, il y a toute la partie pour que les cartes se retournent quand on passe dessus. Voici donc tout le code CSS pour cette partie :

```
.flip-card {
  background-color: transparent;
  perspective: 1000px;
  border: 1px solid #f1f1f1;
}
.flip-card-inner {
  display: grid;
  width: 100%;
  height: 100%;
  text-align: center;
  transition: transform 0.8s;
  transform-style: preserve-3d;
}
.flip-card:hover .flip-card-inner{
  transform: rotateY(180deg);
}
.flip-card-front, .flip-card-back{
  grid-area: 1/1;
  width: 100%;
  height: 100%;

  -webkit-backface-visibility: hidden;
  backface-visibility: hidden;
}
.flip-card-front{
  background-color: white;
  color: black;
}
.flip-card-back{
  background-color: white;
  color: black;
  transform: rotateY(180deg);
}
```

Les deux éléments en gras permettent de superposer l'avant et l'arrière de la carte et pas qu'ils soient décalés.

### 21.3.2 Javascript

Dans le javascript, il n'y aura pas vraiment de code spécial mis à part pour la carte où l'on devra utiliser la fonction hover. Voici donc la fonction hover qui permet de récupérer le nom du compositeur sur lequel on se situe :

```
$(".flip-card").hover(  
  function () {  
    let val = "";  
    for (const compo in data.composers) {  
      if (data.composers[compo].complete_name ===  
(this).find(".flip-card-inner .flip-card-front p").text()) {  
        val = data.composers[compo].id;  
      }  
    }  
    let cmp = new BackComposspecifiqueCtrl(val);  
  },  
  function () {}  
);
```

### 21.3.3 Appel au webService

Et pour finir voici un exemple de mes appels webService avec un paramètre dans la méthode :

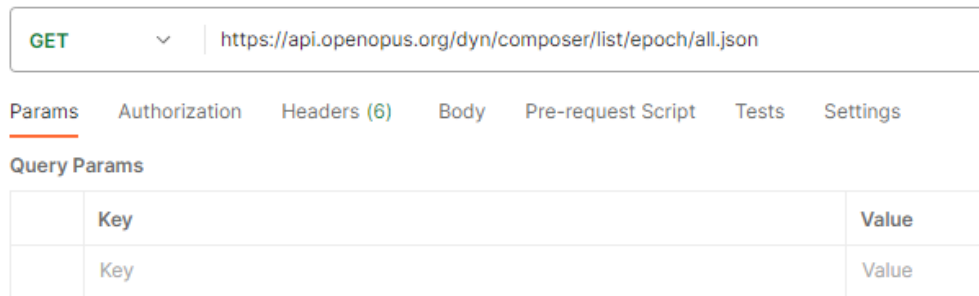
```
chercherCompoSpecifique(compositeur, successCallback) {  
  let url = "https://api.openopus.org/dyn/work/list/composer/" +  
  compositeur + "/genre/all.json";  
  // envoi de la requête  
  $.ajax(url, {  
    type: "GET",  
    contentType: "application/x-www-form-urlencoded; charset=UTF-8",  
    success: successCallback  
  });  
}
```

## 21.4 Tests

Dans la partie des tests, il y aura deux parties distinctes. La première partie est les tests que l'on va faire au début sur postman pour tester notre API. Et la deuxième partie est les tests sur le site final.

### 21.4.1 Postman

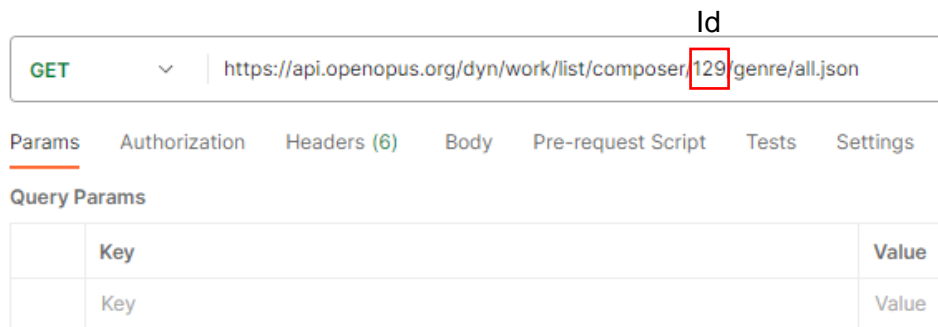
Pour la première partie, on va donc avoir trois requêtes à tester. Une pour récupérer tout les compositeurs, un pour un compositeurs spécifique et une pour un morceau. Pour la première, voici la requête que l'on va faire :



The screenshot shows a Postman interface for a GET request. The URL bar contains 'https://api.openopus.org/dyn/composer/list/epoch/all.json'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Query Params', there is a table with two rows, each with 'Key' and 'Value' columns.

Key	Value
Key	Value

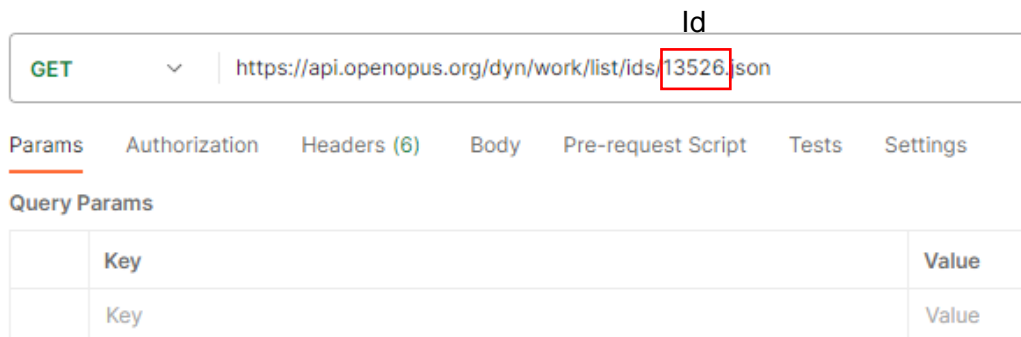
Ensuite pour la deuxième requête, on va choisir un id d'un compositeur au hasard et voici la requête que l'on aura :



The screenshot shows a Postman interface for a GET request. The URL bar contains 'https://api.openopus.org/dyn/work/list/composer/129/genre/all.json'. The number '129' is highlighted with a red box. Above the URL bar, the word 'Id' is written. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Query Params', there is a table with two rows, each with 'Key' and 'Value' columns.

Key	Value
Key	Value

Et pour la dernière requête, on va aussi choisir un id de morceau au hasard :



The screenshot shows a Postman interface for a GET request. The URL bar contains 'https://api.openopus.org/dyn/work/list/ids/13526.json'. The number '13526' is highlighted with a red box. Above the URL bar, the word 'Id' is written. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Query Params', there is a table with two rows, each with 'Key' and 'Value' columns.

Key	Value
Key	Value

### 21.4.2 Fonctionnement sur le site

Et pour la deuxième partie, on va tester les différentes fonctionnalités du site web. Voici le résultat du protocole de tests :

No	Description	Résultat attendu	Résultat obtenu
1	Sélection d'un élément dans le filtre	Seuls les compositeurs du filtre sont affichés	OK
2	Passer par-dessus un compositeur	La carte doit se retourner et afficher des infos sur le compositeur	OK
3	Cliquer sur le compositeur	La liste des morceaux s'affiche sur la droite	OK
4	Cliquer sur un morceau	Le détail du morceau s'affiche sur la droite	OK

## 21.5 Hébergement et fonctionnement

Pour l'hébergement de ce site web, nous allons le mettre sur le dépôt prêté par l'EMF sous ce lien : <https://carrardr.emf-informatique.ch/307/Projet/>

Pour la documentation, elle se situe sur mon github (comme déjà dit précédemment) mais voici de nouveau le lien : [Compositeur\\_project\\_api/code at main · CarrardR/Compositeur\\_project\\_api \(github.com\)](https://github.com/CarrardR/Compositeur_project_api)

Pour le fonctionnement de ce site web, nous arrivons sur un page qui contient tout les compositeurs. Sur la gauche on a à disposition un filtre pour choisir l'époque des compositeurs. Si on passe par-dessus un compositeur, le compositeur va se tourner et des informations sur lui vont s'afficher. Si on clique dessus le compositeur, on va avoir à droite la liste des morceaux qui va s'afficher. Et si on clique sur un des morceaux, des détails sur le morceau seront affichés.

## 22 Conclusion

Et pour terminer ce rapport, voici la conclusion. C'était un module qui nous a permis de revoir un petit peu la programmation web étant donné que cela faisait plus d'un an que nous n'en n'avions pas fait. Et j'ai trouvé que c'était intéressant de s'intéresser pas seulement à l'affichage du site mais d'aller fouiller dans les fonctionnalités que l'on peut ajouter dans notre site grâce à javascript. Au début c'est un peu compliqué de comprendre la structure pour les appels vers une API mais dès qu'on a compris c'est simple.

Ce que j'ai moins aimé dans ce module c'est que on a souvent des erreurs et on ne sait pas pourquoi, et c'est souvent juste un élément qui est mal écrit ou comme ça.

Et sinon pour ce que j'ai aimé, je dirai que c'est quand on fait des exercices et que l'on a le résultat que l'on veut. J'ai bien aimé aussi la partie projet et découvrir des nouvelles fonctionnalités.

L'ambiance de la classe était aussi favorable au travail parce qu'il n'y avait pas trop de bruit et on arrivait à se concentrer.