Web avancé – L2 MIASHS

TD 04 - Javascript, manipulation du DOM et Promises !

La manipulation du DOM en Javascript est très importante pour profiter de la flexibilité d'affichage que permet le langage. Dans ce TD nous allons encore fois manipuler le DOM par Javascript, mais aussi découvrir comment un simple affichage aléatoire d'images avec animations de transition vous oblige à manipuler le DOM, gérer le timing et utiliser le CSS de manière adéquate.

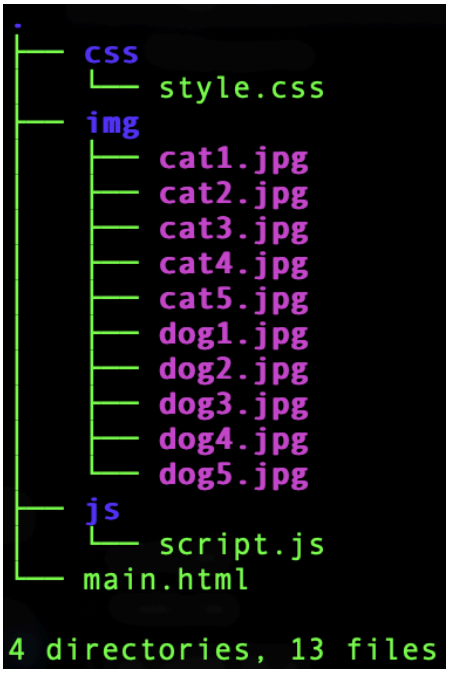
Objectif : consolider la manipulation du DOM par Javascript, découvrir les promesses
Cible : les étudiants ayant les bases de Javascript mais souhaitant consolider leur manipulation du DOM et découvrir les Promises
Durée : très variable, en fonction de votre niveau de départ. Quoi qu'il en soit, essayez de vraiment finir ce TD ! Même chez vous.

Chaque exercice se distingue par ce logo !
Dans ce TD, le code est intégré sous forme d'image afin que vous ne puissiez pas simplement le copier-coller.

Mise en place de la structure

Avant de commencer à vraiment faire du Javascript, il convient de mettre en place toute la structure de l'application. Pour ce faire :

- Créez la structure de dossiers et fichiers suivante :



Pour cela, téléchargez les images à l'adresse suivante :
https://gitlab.com/gguibon/idmc-storage/-/raw/main/webl2/img_td04.zip
Puis décompressez le dossier et mettez les images au bon endroit (dans un dossier **img** donc).

Mise en place de l'HTML de base (main.html)

Bien que la majorité du code HTML viendra du Javascript, il est nécessaire de créer le fichier HTML de base.
Pour cela, dans **main.html** :

- Créez le code HTML initial avec un encodage utf-8 et un titre d'onglet " 🐶🐱"
- Liez ce code HTML avec le fichier **style.css**
- Dans le body, créez une balise **div** ayant pour la classe "content"
- Dans cette même balise div, créez un titre de niveau 1 ayant pour contenu "Animaux Meugnons 🥰" et une balise **div** ayant pour identifiant "viewpanel"
- Liez ce code HTML avec le fichier **script.js**

Mise en place du CSS de base (style.css)

Passons désormais au CCS nécessaire pour notre affichage. Ce code CSS n'a pas grand chose de nouveau par rapport au CSS que vous avez fait précédemment, mais peut servir de rappel.. À vous de mettre en place les éléments suivants :

Positionnement des éléments sur la page :

- Créez une classe **column** qui sert à positionner l'élément en flottant (float) sur la gauche.
- Créez une classe **col-100** qui donne une largeur de 100% (width)

Style des boutons :

- Créez la classe **clickable** qui change le curseur en **pointer**
- Créez la classe **primary** qui donne une couleur de fond bleue (#3366ff) et une couleur de texte blanche
- Faites en sorte que passer le curseur de la souris sur un élément ayant la classe primary change la couleur de fond en bleu plus foncé (#0843f5)
- Créez une classe **button** ayant
    - aucune bordure
    - des coins arrondis à 5 pixels (rayon de la bordure)
    - un rembourrage (padding) vertical de 15 pixels et horizontal de 32 pixels
    - un texte centré (text-align)

  - un affichage de type inline-block
  - une taille de police de 16 pixels
  - une marge (margin) verticale de 4 pixels et horizontale de 2 pixels
  - un curseur de type pointer
  - aucune décoration de texte

L'objectif sera d'obtenir le résultat suivant pour les boutons :

Style pour la carte :

- Créez une classe **card** ayant
  - une ombre (box-shadow) avec un offset-x de 0 pixels, un offset-y de 4 pixels, un rayon de floutage (blur-radius) de 8 pixels, et un spread-radius de 0 pixels. Cette ombre doit être de couleur noire avec une opacité de 20%, donc rgba(0,0,0,0.2)
  - des coins arrondis à 5 pixels (border-radius)
  - un texte centré (text-align)
  - une largeur de 100% (width)
  - une marge (margin) générale de 5 pixels
- Créez une classe **container** donnant un rembourrage (padding) vertical de 2 pixels et horizontal de 16 pixels
- Créez une classe **avatar** qui servira à donner du style aux images avec
  - des coins arrondis à 5 pixels (border-radius)
  - une hauteur maximale de 400 pixels (max-height)
- Créez une classe **content** pour contrôler la position du contenu
  - avec une marge haute de 50 pixels (margin-top)

Animations CSS :

Nous voulons deux animations, une pour l'apparition qui va rendre la carte visible progressivement et grande progressivement.

- Créez une animation nommée **apparition** qui part d'une échelle de taille (scale) égale à 0 et d'une opacité (opacity) de 0% ; puis qui finit sur une taille (scale) de 1 et une opacité (opacity) de 100%
- Créez une classe **apparition** qui contient l'animation **apparition** pour 1 seconde.
- Faites l'inverse de cette animation et de cette classe pour une animation disparition et une classe disparition.

Javascript : Création de la carte en mémoire

Tout est mis en place pour enfin pouvoir uniquement nous concentrer sur le Javascript dans **script.js**. Nous allons commencer par pouvoir bien créer en mémoire la carte. Mais pour cela :

- Créez une fonction qui accepte en argument un chemin d'image et créer en mémoire la structure suivante.

```
<div class="row apparition">
    <div class="column col-100">
        <div class="card">
            <img class="avatar" src="img/cat2.jpg" alt="chat" id="animalImage">
            <div class="container">
                <h4>chat mignon</h4>
                <button class="button primary">Changer</button>
            </div>
        </div>
    </div>
</div>
```

(“img/cat2.jpg” doit évidemment être issu du contenu de l'argument de la fonction)

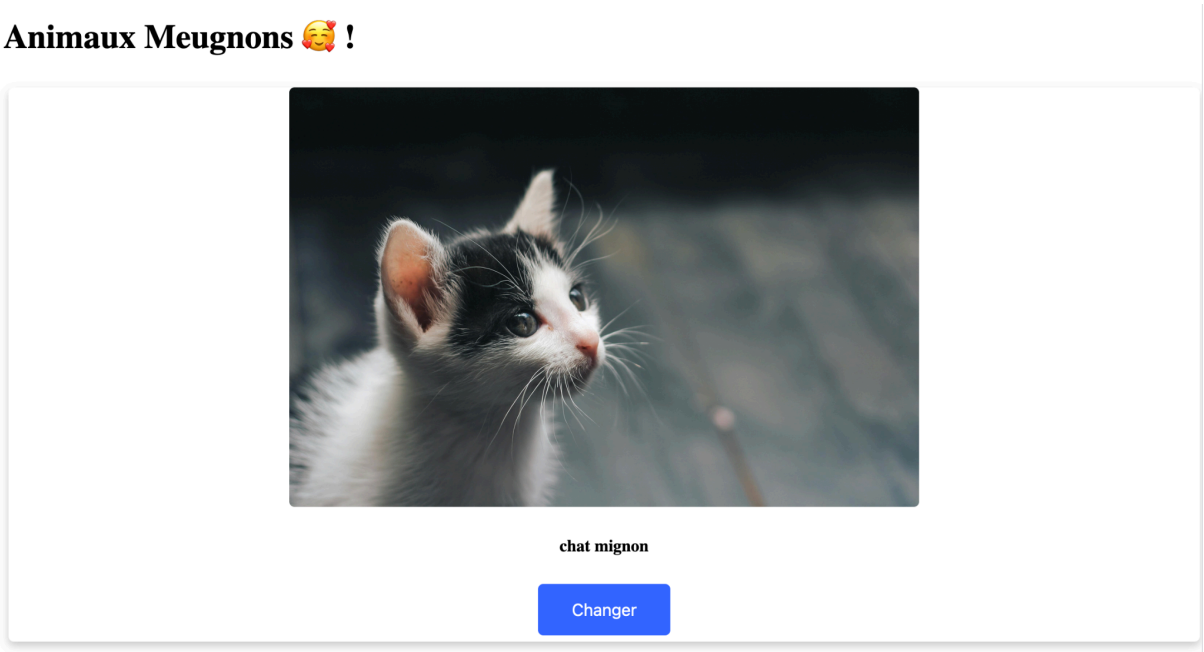
- “chat” et “chat mignon” doivent être conditionnés par le chemin de l'image. Pour cela, créez une fonction qui va retourner “chat” ou “chien” en fonction du chemin relatif de l'image. Utilisez-la ensuite dans la fonction précédente pour les valeurs du titre de niveau 4 et de l'attribut alt.

Javascript : Image aléatoire et attachement au DOM

Nous avons ce qu'il faut pour créer en mémoire la carte. Cependant, il reste à l'attacher au DOM mais aussi à sélectionner aléatoirement une image parmi celles que nous possédons. Pour ce faire :

- Créez une variable globale contenant un tableau de tous les noms de fichiers (cat et dog).
- Créez une fonction qui utilise cette variable globale pour y sélectionner un nom aléatoire et rendre le chemin relatif de l'image portant ce nom.
    - 💡utilisez pour cela la concaténation de Strings
- Créez une fonction qui va attacher cette carte au DOM par le biais de l'élément ayant l'identifiant “viewpanel”
- Créez enfin une fonction qui va servir à afficher un animal aléatoire. C'est-à-dire que cette fonction va réutiliser les fonctions précédentes pour
    - obtenir un chemin relatif d'une image aléatoire
    - créer un l'HTML de la carte d'un animal en mémoire
    - attacher ce contenu HTML dans le DOM
- Appelez cette dernière fonction directement.

Vous pouvez enfin ouvrir votre fichier HTML dans votre navigateur pour admirer le résultat ! Vous devriez obtenir un résultat comme celui-ci :



Si vous rechargez la page, vous devriez voir l'animation d'apparition de la carte sur une nouvelle image d'animal aléatoire. Vous devriez aussi voir la correspondance entre le texte "chat mignon" et l'image : s'il s'agit de l'image d'un chien, vous devriez voir "chien mignon" inscrit ici.

Javascript : événement, promesse et animation de disparition

Certes, la page est super, le chat ou le chien est vraiment trop craquant ! Mais !!! Quid du bouton présent ? À quoi sert-il ? 😅

L'objectif est de faire en sorte que cliquer sur ce bouton active l'animation de disparition de la carte et génère une nouvelle carte qui apparaît à la place par l'animation d'apparition. Cela peut paraître simple dit comme ça, mais cela signifie de devoir gérer de manière asynchrone les différentes actions !

Pour cela, nous devons d'abord créer une promesse (promise) :

- Créez une fonction qui va attendre un certain nombre de millisecondes données à argument
    - Cette fonction retourne une nouvelle instance de la classe Promise. Elle active la fonction **resolve** pour indiquer la fin des actions asynchrones. Pour cela, adaptez le code suivant, pour attendre un certain nombre de

millisecondes à l'aide la fonction setTimeout().

```
function waitMilliseconds(ms) {
    return new Promise((resolve) => {

        // faire des choses

        // indique que c'est fini
        resolve('fini');

    });
}
```

Maintenant que nous avons une promesse permettant d'avoir un retour quand l'action parallèle est finie, il convient d'utiliser une fonction asynchrone pour gérer l'attente de la fin de l'action asynchrone (à l'aide de la promesse et de notre utilisation de **resolve('fini')**).

Pour cela :

- Créez une fonction asynchrone qui va attendre (await) que la fonction précédente soit finie pour supprimer le contenu HTML de l'élément ayant l'identifiant "viewpanel" puis afficher (et attacher au DOM donc) un nouvel animal (à l'aide de la fonction déjà créée à cet effet).

Enfin, il manque l'activation de l'animation de disparition et l'attente pour remplacer la visualisation lors du clic sur le bouton.

Pour cela :

- Modifiez votre fonction de création de l'HTML en mémoire pour y ajouter un écouteur d'événement (eventListener) sur le click qui
    - enlève la classe CSS "apparition" de toute la carte (et la rangée)
    - ajoute la classe CSS "disparition"
    - appelle la fonction asynchrone pour attendre 800 millisecondes et afficher un nouveau animal

Vous devriez désormais pouvoir tester et vérifier la manipulation du DOM avec une animation de disparition puis d'apparition d'une carte avec un nouvel animal aléatoire.

Encore plus loin : empêcher la sélection aléatoire du même animal deux fois de suite

Souvent, le même animal va revenir. Il nous faut empêcher cela.

Pour ce faire, modifiez le code en conséquence.
💡 Je vous conseille de modifier la fonction asynchrone et la fonction d'obtention de l'image aléatoire
💡 Vous aurez surement besoin de la création d'une copie réelle du tableau à l'aide de Array.from()

C'est bon pour ce TD ! Youhou ! Vous en êtes venu à bout !

Ce TD vous a permis de consolider votre manipulation du CSS, des animations CSS, du Javascript avec manipulation du DOM. Il vous a également donné un premier contact avec les fonctions asynchrones et les promesses, qui seront très importantes pour le prochain TD sur la manipulation de l'AJAX ! 😉

Liens utiles

- https://developer.mozilla.org/fr/docs/Web/JavaScript
- https://www.w3schools.com/js/default.asp