Partie 4: Formulaire avec gestion du texte et des images

Dans cette partie, nous allons nous concentrer sur comment ajouter un film.

Manipulation du DOM 1/2

Petit point théorique sur la gestion des événements du DOM.
Nous travaillons en VanillaJS (= en JavaScript pur, sans framework, sans librairie) cela veut dire que pour simplement et efficacement manipuler le DOM nous utilisons:

```
document.querySelector("#img").addEventListener("click", (event) => {
  console.log(42);
});
//Traduction de ces quelques lignes: au clic sur l'élément du DOM portant l'id "img",
//je fais sortir dans ma console navigateur le nombre "42"
```

Vous serez peut-être d'accord avec moi, écrire toutes ces lignes à chaque fois qu'on veut manipuler un bout du DOM ça devient assez pénible.
Nous allons donc créer ensemble notre première fonction de notre PWA. Cette fonction va s'appeler *DomOn*. La voici:

```
const domOn = (selector, event, callback) => {
  document
    .querySelectorAll(selector)
    .forEach((el) => el.addEventListener(event, callback));
};
```

QUESTION: Que pouvez-vous dire sur cette fonction, que fait-elle ?

Cette fonction qui prend 3 paramètres en entrée va tout d'abord sélectionner un élément du DOM.

POINT THÉORIQUE: la fonction *querySelectorAll* retourne une liste de nœuds (NodeList) HTML qui matche le sélecteur (selector). Il y en a donc potentiellement plusieurs (si le sélecteur est une classe par exemple).

C'est pour cela que l'on va itérer dessus pour choper tout les éléments du DOM qui matche le sélecteur. Le tout grâce au *forEach*.

Dans ce *forEach* chaque élément (*el*) sera lié à un *EventListener* qui prend en argument l'événement en soi et la fonction de callback (= l'action à faire lors du match entre le sélecteur et l'événement).

Pour utiliser cette fonction, on fera donc:

```
domOn("monSélecteur", "monÉvénement", (evt) => {
  //ma fonction de callback
  // je fais ce que je veux ici
});
```

Cette orthographe ressemble assez à celle de JQuery. C'est vrai ! Et c'est exactement le point important ! Puisque JQuery reste une libraire JS, il n'y a que du JavaScript derrière. Nous venons donc de démontrer à une petite échelle ce que fait JQuery quand vous appelez le *$*.

Ressources

querySelectorAll Doc MDN

AddEventListener Doc MDN

Exercice pratique n°2

DONNÉE: le but de cet exercice est de créer un formulaire. Veuillez donc créer une nouvelle *section* dans votre HTML. Attention, pas une nouvelle page, car nous sommes en Single Page App !
Puis dans cette nouvelle section, créez un formulaire où il sera possible de renseigner les champs décrits ci-dessous:

- Le titre du film
- La date de visionnage du film
- Une image du film
- Les étoiles (la note) attribuées au film

ATTENTION: Nous sommes en SPA, faites le nécessaire pour que la section d'ajout d'un film ne soit pas visible au chargement de la page, mais seulement lors du clic sur le bouton *+* de la section *global-page*, codé dans l'EX1.

Manipulation du DOM 2/2

Nous sommes bien d'accord (je l'espère) ajouter à la main chaque film directement dans le HTML, c'est un peu contre-productif. Nous allons donc voir le sauveur de nos nuits de code: le templating !

Le template traduit de l'anglais => "modèle" est une façon de manipuler le DOM en ayant un modèle (la forme) où nos données (le fond) vont se positionner.

Nous pourrions utiliser des librairies de templating tel que Handlebars. Mais nous codons en VanillaJS donc pas de librairies inutiles qui prennent de la place et dont nous devons apprendre le fonctionnement. Comme JQuery avant, Handlebars est construit en JS, nous pouvons donc faire la même chose en JS pure. C'est donc ce que nous allons faire pour bien comprendre comme cela marche.

```
const template = ({
  template: OnWorkingTemp,
  selectors: selectors,
  vars: vars,
}) => {
  const tempToDress = OnWorkingTemp.querySelectorAll(selectors);
  for (const part of tempToDress) {
    Object.keys(vars).forEach((variable) => {
      if (part.textContent === variable) {
        part.textContent = vars[variable];
      }
    });
  }
  return OnWorkingTemp;
};
```

QUESTION: Que pouvez-vous dire sur cette fonction, que fait-elle ?

Cette fonction va prendre 3 paramètres

POINT THÉORIQUE: vous pouvez nommer vos paramètres en ajoutant des *{ }* dans votre fonction, comme dans cette fonction. Cela va nous permettre de pouvoir envoyer les paramètres dans le désordre quand nous utiliserons la fonction, mais d'être sûrs qu'il soit mis à la bonne place.

```
//utilisation
const templateDone = template({
  template: clone,
  vars: vars,
  selectors: "p, span",
});
```

Pour revenir sur ce que fait la fonction; elle va prendre le paramètre *template* qui est un clone d'un nœud du HTML. Ce clone est une simple ligne de JS qui va, comme son nom l'indique cloner une partie de l'HTML.

```
const TEMP_MOVIE = document.querySelector("#temp-movie"); // je prends un bout de l'HTML

const clone = TEMP_MOVIE.content.cloneNode(true); // je le clone pour travailler sur une copie
//ATTENTION il faut cloner à chaque fois que l'on veut utiliser le template
```

Concentrons-nous sur le bout de HTML qui va être cloné et introduisons une nouvelle balise HTML:

```
<template> </template>
```

- Tout ce qui ira entre cette balise ne sera pas visible lors du chargement de la page.
- Il pourra être affiché par un script JS par la suite.

Pour ce qui est du paramètre *vars*, elle va tous simplement prendre un objet de JS ayant une clef et une donnée.

```
const vars = {
  title: "le titre de mon film",
  day: "le jour de visionage en chiffre",
  month: "le mois de visionage en lettre",
  year: "l'année de visionage en chiffre",
};
```

Pour ce qui est du paramètre *selectors*, elle va tous simplement prendre les sélecteurs envoyés pour pouvoir choper à l'interne du clone les éléments dont elle a besoin.

Nous sommes arrivés à la fin de démantèlement de la fonction présentée en slide 8, remontons maintenant le chemin grâce à quelques petits ex-pratiques pour en comprendre le fonctionnement.

Ressources

<template> Doc MDN

Exercice pratique n°3 A

DONNÉE: Transformer votre partie d'HTML qui représente un film en un template.

- Il faut réussir à prendre la structure (la forme) du HTML et en faire un template global.
- À la place de la donnée (le fond) présente entre les balises du HTML (comme par exemple le titre du film), remplacez celle-ci par un nom représentant le sens de cette donnée (par exemple pour le titre du film au lieu de: *Grave* => *title*).
- Supprimer les autres parties du HTML que vous avez copié-collé dans votre code.
- Garder simplement une *div* globale où vous pourrez mettre tous vos films.
- **ATTENTION** les sélecteurs que vous enverrez à la fonction doivent être les mêmes que ceux que vous utiliser dans votre template HTML

SOLUTION:

```
<!-- templates -->
    <template id="temp-movie">
        <div class="content-image">
            <button id="delete">&#10005</button>//code HTML représentant une "x" (https://unicode-table.com/fr/002B/)
            <img class="image" src="" alt="">
            <div class="content">
                <p class="title">title</p>
                <span class="watching-date">
                    <span class="day">day</span>
                    <span class="alphabetical-month">month</span>
                    <span class="year">year</span>
                </span>
                <span class="rating"></span>
                //nous verrons ici comment faire un code CSS pour mettre directement les étoiles
            </div>
        </div>
    </template>
<!-- end of templates -->
```

Exercice pratique n°3 B

DONNÉE: Copier-coller la fonction de templating de la slide 8 et utilisez-la pour créer le code HTML d'**UN** film.

TIPS: Ce que vous recevez après avoir appelé la fonction *template* est un bout de DOM, **il est donc manipulable comme un bout de DOM ordinaire**

RESOURCES: toutes les réponses ne sont pas dans les slides, mais ceci peut vous aider:

- appendChild
- pour gérer l'image

SOLUTION:

```
const TEMP_MOVIE = document.querySelector("#temp-movie");
const clone = TEMP_MOVIE.content.cloneNode(true);

const vars = {
  title: "Grave",
  day: 15,
  month: "Juillet",
  year: 2018,
};

const templateDone = template({
  template: clone,
  selectors: "p, span",
  vars: vars,
});
templateDone.querySelector(".image").src = "./assets/grave.jpg";
document.querySelector("#home").appendChild(templateDone);
```

Month To Month

Un petit aparté fait toujours du bien !

N'avez-vous par remarqué que quand vous allez interagir avec le formulaire, la date que vous allez recevoir sera sous un format, comment dire... pas comme vous voudrez ?

Nous allons corriger ce problème ! Grâce à une fonction ! (vive les fonctions ❗)

```
const months = {
  1: "Janvier",
  2: "Février",
  3: "Mars",
  4: "Avril",
  5: "Mai",
  6: "Juin",
  7: "Juillet",
  8: "Août",
  9: "Septembre",
  10: "Octobre",
  11: "Novembre",
  12: "Décembre",
};

const monthToMonth = (date) => {
  const JSdate = new Date(date);
  const day = JSdate.getDate();
  const month = JSdate.getMonth() + 1;
  const year = JSdate.getFullYear();

  const alpahbeticalMonth = months[month];

  const finalDate = {
    day: day,
    month: alpahbeticalMonth,
    year: year,
  };

  return finalDate;
};
```

QUESTION: Que pouvez-vous dire sur cette fonction, que fait-elle, que renvoie-t-elle ?

Exercice pratique n°3 C

DONNÉE: Récupérer les données du formulaire pour les insérer par la suite dans le template

- L'image n'est pas à prendre en compte actuellement
- Attention ce n'est pas du PHP, mais bien du JS en SPA que nous faisons, donc le formulaire n'est pas envoyé, mais les données doivent quand même être récupérées 😄

TIPS:

- n'oubliez pas de faire des *console.log()* le plus souvent possibles pour voir ce que vous retournent vos actions / votre code
- le nom présent dans le template (*day* day ici) doit être le même que la clef dans l'objet JS qui envoie les données au template.

RESOURCES: aucune réponse n'est dans les slides, à vous d'aller chercher parmi la doc.

- Formulaire (pour vous aiguiller)