



**POLYTECHNIQUE  
MONTREAL**

UNIVERSITÉ  
D'INGÉNIERIE

# LOG2440 – Méthod. de dévelop. et conc. d'applic. Web

## Travail pratique 2

Chargés de laboratoire:

Thierry Beaulieu

Rachad Chazbek

Charles De Lafontaine

Jamesley Joseph

Antoine Poellhuber

Automne 2022

Département de génie informatique et génie logiciel

# 1 Objectifs

Le but de ce travail est de vous introduire au langage de programmation JavaScript et la manipulation du DOM (Document Object Model). Vous allez vous familiariser avec la syntaxe ES2015 de JavaScript et la gestion des événements. Finalement, vous utiliserez un outil d'analyse statique pour le respect de conventions de programmation (*linter*) : ESLint. Cet outil définit un ensemble de règles à suivre pour un code uniforme et standard.

De manière plus concrète, vous aurez à mettre en place plusieurs interactions sur différentes pages du site web afin que celui-ci devienne dynamique et fonctionnel. À la fin de ce travail, le site web sera utilisable, mais aucune mesure de sécurité ne sera présente puisque l'ensemble des opérations seront réalisées du côté client.

## 2 Introduction

Lors du premier travail pratique, vous avez mis en place la structure et la mise en forme du site web. Cependant, celui-ci était complètement statique. En effet, il n'était pas possible de choisir une playlist spécifique à afficher ou d'interagir avec le contenu du formulaire ou de faire jouer une chanson à travers les contrôles de musique.

Afin d'être en mesure d'ajouter des interactions du côté client, le langage JavaScript doit être utilisé. En ce sens, ce langage permet entre autres de manipuler les éléments du [DOM](#) (*Document Object Model*), d'avoir accès aux API du navigateur (ex : [Local Storage](#)). Dans le cadre de ce travail pratique, la syntaxe *moderne* du JavaScript, ES2015, sera utilisée.

## 3 Configuration du projet

### 3.1 Modules ECMAScript (ESM) et serveur HTTP local

Au cours de ce travail, vous aurez à utiliser les modules ESM (*ECMAScript*). Ces modules doivent obligatoirement être servis par un serveur HTTP. Dans votre cas, vous utiliserez [lite-server](#), un serveur HTTP statique local qui fournit les fichiers de votre travail. Pour lancer le serveur, vous devez utiliser la commande **npm start** dans votre terminal. Assurez-vous d'avoir installé les dépendances du projet avec la commande **npm ci** au préalable.

Par défaut, ce serveur est accessible à l'adresse : "http://localhost:5000".

**Note :** le nom **localhost** est un raccourci pour l'adresse IP **127.0.0.1**. Le lancement du serveur *lite-server* affichera les adresses auxquelles le serveur est accessible.

## 3.2 Tests automatisés

Comme pour le premier travail, plusieurs tests vous sont fournis dans le répertoire `tests/e2e`. Ces tests vous permettront de valider votre travail et le contenu de vos pages HTML. Il est fortement recommandé de lire les tests et leur description avant d'écrire votre code pour vous aider avec le résultat final.

Dans ce travail, vous aurez un ensemble de tests supplémentaires qui viendra s'ajouter aux tests du premier travail pratique. Ces tests vous permettront de valider certaines fonctionnalités supplémentaires à ajouter dans ce travail pratique.

Pour lancer la suite de tests, vous n'avez qu'à aller à la racine de votre répertoire git avec un terminal et de lancer la commande **npm run e2e** afin de lancer les tests fournis. Évidemment, vous devrez au préalable avoir installé l'environnement d'exécution [NodeJS](#).

**Note : comme le site web utilise ESM, il faudra valider des pages HTML fournies par un vrai serveur. Il faut obligatoirement avoir lancé le serveur statique avec la commande `npm start` avant de lancer les tests avec `npm run e2e`. Voir la section 3.1.**

## 4 Travail à réaliser

En vous basant sur le travail pratique précédent, vous aurez à ajouter la couche de logique du site web grâce au langage JS. Un corrigé du TP1 vous sera fourni avec le dépôt de Git de départ afin de partir du bon pied pour ce travail. Notez que certaines pages HTML peuvent avoir été légèrement modifiées par rapport à l'énoncé du TP1 pour les besoins du TP2.



### Avertissement

---

Afin de promouvoir la compréhension du langage JavaScript, les bibliothèques telles que jQuery ne sont pas permises pour ce travail pratique. Vous devez manipuler le DOM seulement à travers les méthodes natives.

---

### 4.1 Éléments à réaliser

Avant de débiter, assurez-vous que le projet de départ est fonctionnel et vous êtes capables de naviguer à travers les pages du document. Le code de départ présente un affichage statique des pages. Vous devez enlever certaines parties du HTML puisqu'elles seront générées par le code JS. Ces parties sont clairement indiquées par des commentaires dans les fichiers HTML.

## 4.2 Chargement des données

Les fichiers `playlists.js` et `songs.js` contiennent les playlists et chansons de départ pour le travail. Afin de pouvoir créer du HTML dynamiquement, votre site doit pouvoir charger ces éléments et les sauvegarder dans son entrepôt local ([Local Storage](#)).

Vous devez compléter la classe `StorageManager` qui gère l'accès au *LocalStorage*. Lors du lancement du site, la classe doit vérifier si le storage contient déjà une liste de playlists et une liste de chansons et sinon, charger les fichiers `playlists.js` et `songs.js` et les sauvegarder.

Il doit être possible d'obtenir une des deux listes du storage à travers la méthode `getData` (le code vous est déjà fourni) et sauvegarder une nouvelle playlist ou chanson dans la bonne liste à travers la méthode `addItem`. Vous devez également compléter les méthodes `getItemById` et `getIdFromName` qui permettent de récupérer un item sauvegardé à travers l'attribut `id` ou, inversement, obtenir l'attribut `id` à travers l'attribut `name` de l'élément.

Consultez les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes pour aider à compléter votre code.

*Astuce* : utilisez les méthodes de parcours de tableau pour traiter les listes de données.

## 4.3 Affichage sur la page d'accueil "Ma Bibliothèque"

Votre site doit être capable d'afficher dynamiquement toutes les playlists et chansons disponibles dans le système sur la page principale "Ma Bibliothèque". Le fichier `library.js` implémente le code nécessaire pour la génération dynamique du contenu de la page.

Par défaut, un affichage HTML vous est fourni dans `index.html` pour vous indiquer le code HTML final qui doit être généré. Vous devez retirer le contenu des balises `<section>` ciblées pour permettre au JS de le construire dynamiquement au lancement de la page.

Initialement, les playlists et chansons sont chargées dans le storage. Vous devez récupérer toutes les playlists, incluant celles créées dans la page "Créer Playlist" et toutes les chansons et générer les deux listes et leur HTML dans la page à travers la fonction `generateLists`.

### 4.3.1 Playlists

Vous devez gérer un nombre arbitraire de playlists et générer l'affichage de chaque playlist. Le code de départ fourni présente l'affichage voulu dans l'élément `<section>` ayant l'id `playlist-container`. Vous devez générer le contenu HTML de cet élément avec du code JS.

**Note** : cette balise sera vide à la remise. Le contenu sera généré par le JS.

Vous devez compléter la fonction `buildPlaylistItem` dans le fichier `library.js`. Cette fonction s'occupe de générer le HTML d'affichage pour un item de playlist et le retourner dans un objet `HTMLAnchorElement` (balise `<a>`). Le HTML généré doit être le même que le code de départ et être validé par les tests fournis. La fonction prend en paramètre un objet `playlist` tel que défini dans `playlists.js` et extrait les informations nécessaires. Notez que l'attribut `href` doit pointer vers `./playlist.html?id=x` où `x` est l'attribut `id` de la playlist utilisé lors du chargement de la page `playlist.html`.

Vous devez utiliser les méthodes de manipulation telles que `createElement`, `appendChild`, etc. et non directement l'attribut `innerHTML` pour construire vos éléments HTML.

### 4.3.2 Chansons

Vous devez gérer un nombre arbitraire de chansons et générer l'affichage de chaque chanson. Le code de départ fourni présente l'affichage voulu dans l'élément `<section>` ayant l'id `song-container`. Vous devez générer le contenu HTML de cet élément avec du code JS.

**Note : cette balise sera vide à la remise. Le contenu sera généré par le JS.**

Vous devez compléter la fonction `buildSongItem` dans le fichier `library.js`. Cette fonction s'occupe de générer le HTML d'affichage pour un item de chanson et le retourner dans un objet `HTMLDivElement` (balise `<div>`). Le HTML généré doit être le même que le code de départ et être validé par les tests fournis. La fonction prend en paramètre un objet `song` tel que défini dans `songs.js` et extrait les informations nécessaires.

Notez que les classes utilisées pour le bouton dépendent de l'attribut `liked` de la chanson. Dans le cas qu'une chanson est "aimée", vous devez ajouter la classe `fa`, sinon vous devez ajouter la classe `fa-regular`. Dans les deux cas, les classes `fa-heart` et `fa-2x` sont ajoutées à la liste de classes (attribut `classList`) du bouton.

Vous devez gérer l'événement `click` pour le bouton de chaque chanson. Lorsqu'un utilisateur clique sur le bouton, l'état "aimé" de la chanson doit être inversé. Vous devez gérer le changement d'affichage à travers les classes comme mentionné plus haut. De plus, vous devez mettre à jour l'information dans le storage. La méthode `replaceItem` de la classe `StorageManager` qui vous est fournie vous sera utile.

Vous devez utiliser les méthodes de manipulation telles que `createElement`, `appendChild`, etc. et non directement l'attribut `innerHTML` pour construire vos éléments HTML.

## 4.4 Affichage des information d'une playlist spécifique

Le code à compléter se trouve dans les fichiers `playlist.js` et `player.js`. La classe `PlaylistManager` gère l'affichage de la page et fait appel aux méthodes de la classe `Player` qui gère la logique de contrôle. Consultez les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes pour l'ensemble du fichier.

Pour permettre de savoir quelle playlist charger, un paramètre supplémentaire doit être passé dans l'attribut `href` de chaque balise `<a>`. Ce paramètre correspond à l'attribut "id" de chaque playlist. Par exemple, pour charger la playlist avec `id="3"`, l'URL de redirection doit être `"/playlist.html?id=3"`. L'objet `URLSearchParams` ainsi que les propriétés `search` et `location` de l'objet `document` vous aideront à récupérer la valeur du paramètre `id` de l'URL.

Vous devez générer dynamiquement l'HTML nécessaire pour afficher les chansons de la playlist choisie dans la balise `<section>` ayant l'id `song-container`.

**Note : cette balise sera vide à la remise. Le contenu sera généré par le JS.**

Vous devez utiliser les méthodes de manipulation telles que `createElement`, `appendChild`, etc. et non directement l'attribut `innerHTML` pour construire vos éléments HTML.

Vous devez compléter la méthode `loadSongs` qui charge les chansons de la playlist, génère les éléments HTML représentant les chansons à l'aide de la méthode `buildSongItem` et charge ces chansons dans le `Player` à travers sa méthode `loadSongs`. Vous devez également modifier le nom et l'image de la playlist avec ceux de la playlist choisie.

La méthode `buildSongItem` est similaire à celle dans `library.js`. La différence est que vous n'avez pas à gérer le changement de l'état "aimé" d'une chanson, mais vous devez gérer l'événement `click` sur l'ensemble de l'élément HTML retourné. Lorsqu'un utilisateur clique sur cet élément, la chanson est jouée à travers `playAudio` et l'information de la chanson en cours est mise à jour à travers `setCurentSongName`.

La méthode `setCurentSongName` met à jour l'élément `id="now-playing"` avec le nom de chanson en cours. Cette méthode est appelée chaque fois que la chanson en cours change.

Les méthodes `playAudio`, `playPreviousSong` et `playNextSong` interagissent avec la classe `Player` et permettent de changer la chanson en cours. La méthode `playAudio` permet également de mettre en pause/jouer la chanson. Lorsqu'une chanson est mise en pause, le bouton `play` doit avoir la classe `fa-play`, sinon il doit avoir la classe `fa-pause`.

La méthode `shuffleToggle` permet d'activer/désactiver le mode *shuffle* de la classe `Player`. Lorsque le mode est activé, le bouton `shuffle` doit avoir la classe `control-btn-toggled`, sinon cette classe ne doit pas être présente.

La méthode `muteToggle` permet de fermer/ouvrir le son . Lorsque le son est fermé, le bouton `mute` doit avoir la classe `fa-volume-mute`, sinon il doit avoir la classe `fa-volume-high`.

Les méthodes `audioSeek` et `scrubTime` interagissent avec la classe `Player` et permettent de modifier la position dans la chanson en cours à travers la manipulation de la barre de progrès ou l'utilisation de raccourcis pour avancer/reculer de quelques secondes. Dans les deux cas, la logique doit être gérée par la classe `Player`.

La méthode `bindShortcuts` vous est fournie et configure déjà les raccourcis clavier décrits dans "about.html". Vous devez compléter la méthode `bindEvents` qui configure la gestion des différents événements sur la page. Consultez les commentaires dans le code fourni pour savoir quel événement gérer et quelle action effectuer pour sa gestion. Les deux méthodes doivent être appelées lors du chargement de la page.

#### 4.4.1 Player

La classe `Player` dans `player.js` gère la logique du son à travers un objet [Audio](#). Plusieurs méthodes de `PlaylistManager` font appel à cette classe tel qu'expliqué dans la section 4.4. Vous devez compléter les méthodes marquées par un **TODO** dans leur en-tête.

La classe garde une référence vers la chanson en cours à travers un index dans son attribut `currentIndex`. Cet index représente la position de la chanson en cours dans le tableau `songsInPlayList`. L'ordre des chansons se comporte comme une boucle infinie, c.-à.-d que lorsque la dernière chanson est jouée, la prochaine chanson à jouer sera la première du tableau. La chanson qui précède la première du tableau est la dernière du tableau.

La méthode `loadSongs` permet de charger les chansons à jouer et attribue par défaut la première chanson comme source d'audio (attribut `src` de l'objet `audio`).

La méthode `playAudio` permet de jouer une chanson en fonction de son index (voir fichier `song.js`). Pour modifier la source de l'audio, il faut modifier son attribut `src`. Référez-vous à la documentation de [HTMLMediaElement](#) pour les méthodes et attributs à modifier pour jouer du son. Si l'index passé en paramètre a la valeur `-1`, la fonction met en pause/joue la chanson en cours en fonction de l'attribut `paused` de l'objet `audio`.

Les méthodes `playPreviousSong` et `playNextSong` permettent de jouer la chanson d'après ou d'avant dans la liste. Ces méthodes doivent tenir compte du comportement de boucle

infinie expliqué plus haut pour identifier la prochaine chanson à jouer. La fonction `modulo` dans `utils.js` peut être utile dans ce cas.

Si l'attribut `shuffle` est `true`, la prochaine chanson est choisie de manière aléatoire. La fonction `random` dans `utils.js` peut être utile dans ce cas. Notez qu'une implémentation de base de l'aléatoire est suffisante : il se peut qu'une chanson se répète dans le mode "shuffle".

Les méthodes `muteToggle` et `shuffleToggle` permettent d'activer/désactiver les modes "mute" et "shuffle". Dans les deux cas, le nouvel état est retourné par la fonction.

La méthode `scrubTime` permet de modifier le moment de l'audio joué à travers l'attribut `currentTime` en lui ajoutant un temps en secondes. Notez que le temps `delta` peut être également une valeur négative.

## 4.5 Ajouter une nouvelle playlist

Votre site doit être permettre à l'utilisateur d'ajouter sa propre playlist à travers le formulaire de la page "create\_playlist.html". Une playlist ajoutée à travers le formulaire est sauvegardée localement et disponible dans la liste de la page "Ma Bibliothèque". Le code à compléter se trouve dans le fichier `playlist_editor.js`. Pour ce TP, seulement la création d'une nouvelle playlist sera implémentée.

Une playlist est représentée par une structure présentée dans le fichier `playlists.js`. Les chansons qui composent la playlist sont identifiées par leur attribut `id` dans le tableau `songs`.

### 4.5.1 Ajouter les choix de chansons

Le menu déroulant représenté par le champ de saisie de type `select` pour chaque chanson utilise un élément `<datalist>` comme source des chansons disponibles. Vous devez compléter la fonction `buildDataList` qui permet de modifier l'élément `<datalist>` et lui ajouter les noms des chansons sauvegardés dans le *LocalStorage* en tant qu'éléments `<option>` enfants.

### 4.5.2 Ajouter une chanson dans la liste

Le bouton `add-song-btn` permet de rajouter une nouvelle section pour l'ajout d'une chanson dans la playlist. Vous devez compléter la fonction `addItemSelect` qui construit le HTML nécessaire. Référez-vous au code fourni dans "create\_playlist.html" et le `<div>` enfant de `<div id="song-list">` pour la structure désirée. L'id de chaque nouvel `<input>` doit être `songs-index` où *index* représente l'ordre des chansons débutant à 1.



Comme on peut vouloir retirer une chanson ajoutée, les éléments `<label>` et `<input>` doivent être accompagnés par un élément `<button class="fa fa-minus">`. Vous devez gérer l'événement `click` sur ce bouton et supprimer le parent `<div>` du DOM. Notez que le champ de saisie initial ne peut pas être retiré et il y a toujours au moins 1 chanson dans la playlist.

Vous devez utiliser les méthodes de manipulation telles que `createElement`, `appendChild`, etc. et non directement l'attribut `innerHTML` pour construire vos éléments HTML.

### 4.5.3 Soumettre le formulaire

Une fois le formulaire rempli, l'utilisateur doit être capable de le soumettre et ajouter la playlist à la liste des playlists sauvegardées localement. Une fois la playlist sauvegardée, elle doit être présente dans la bibliothèque et être consultée comme toute autre playlist.

La soumission du formulaire se fait à travers le bouton "Ajouter la playlist". Vous devez implémenter la gestion d'événement `submit` et rediriger l'utilisateur vers la page "index.html".

Votre code doit récupérer les données du formulaire et construire un objet qui suit la structure d'une playlist. La propriété `elements` d'un formulaire vous sera utile. Lorsque l'objet est construit, vous devez l'ajouter aux autres playlists dans le *LocalStorage*.


La prévisualisation de l'image de playlist choisie vous est fournie. La fonction `getImageInput` vous est fournie pour vous aider à extraire une image à partir d'un champ de saisie de fichier `HTMLInputElement`. Notez que la fonction est marquée comme asynchrone et vous devez donc utiliser le mot clé `await` lors de son appel dans votre code. Exemple : `const monImage = await getImageInput(elements.image).`

Vous devez faire des manipulations supplémentaires pour extraire certaines données. Les chansons sont choisies à travers leur nom, mais la playlist garde des références vers leur attribut `id`. Vous devez alors transformer le nom en un `id` à l'aide de la méthode `getIdFromName` de la classe `StorageManager`. Il n'y a pas de contrainte sur le nombre de fois que la même chanson peut se retrouver dans une playlist : les doublons de chansons sont acceptés.

Chaque playlist possède un `id` unique. La méthode `generateRandomID` dans `utils.js` permet de générer un `id` aléatoire de 10 chiffres par défaut. Notez que contrairement aux chansons dont l'`id` est de type `number`, l'`id` d'une playlist est de type `string`.

## Conseils pour la réalisation du travail pratique

---

1. Lisez **attentivement** l'énoncé, le code fourni et les commentaires dans le code.
  2. Utilisez les outils de développement de votre navigateur web pour vous aider à déboguer votre code JS (raccourci ).
  3. Respectez la convention de codage établie par *ESLint*. Utilisez la commande `npm run lint` pour valider cet aspect.
  4. Utilisez les méthodes des tableaux JS (*map*, *filter*, etc) le plus souvent possible.
  5. Exécutez les tests fournis souvent afin de valider votre code.
- 

## 5 Remise

Voici les consignes à suivre pour la remise de ce travail pratique :

1. Le nom de votre entrepôt Git doit avoir le nom suivant : **tp2\_matricule1\_matricule2** avec les matricules des 2 membres de l'équipe.
2. Vous devez remettre votre code (*push*) sur la branche **master** de votre dépôt git. (pénalité de 5% si non respecté)
3. Le travail pratique doit être remis avant **23h55**, le **23 octobre**.

**Aucun retard** ne sera accepté pour la remise. En cas de retard, la note sera de **0**.

Le navigateur web **Google Chrome** sera utilisé pour tester votre site web.

## 6 Évaluation

Vous serez évalués sur le respect des exigences fonctionnelles de l'énoncé, ainsi que sur la qualité de votre code JS et sa structure. Le barème de correction est le suivant :

Exigences	Points
Code JavaScript	
Code dans <code>storageManager.js</code>	2
Code dans <code>library.js</code>	3
Code dans <code>playlist.js</code>	4
Code dans <code>player.js</code>	3
Code dans <code>create_playlist.js</code>	4
Qualité du code et Respect des règles ESLint	
Structure du code	2
Qualité et clarté du code	2
Total	20

L'évaluation se fera à partir de la page d'accueil du site, soit `index.html`. À partir de cette page, le correcteur devrait être capable de consulter toutes les autres pages de votre site web et interagir avec les différents éléments du site. Tous les tests fournis doivent obligatoirement passer. Le serveur web doit être déployable avec la commande `start` seulement.

Ce travail pratique a une pondération de **6%** sur la note du cours.

## 7 Questions

Si vous avez des interrogations concernant ce travail pratique, vous pouvez poser vos questions sur Discord ou contacter votre chargé de laboratoire.