



**POLYTECHNIQUE  
MONTRÉAL**

UNIVERSITÉ  
D'INGÉNIERIE

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

## Travail pratique 3

Chargés de laboratoire:

Charles De Lafontaine

Antoine Poellhuber

Justin Lachapelle

Hiver 2023

Département de génie informatique et génie logiciel

# 1 Objectifs

Le but de ce travail pratique est de vous familiariser avec les pratiques de tests logiciels, notamment les tests unitaires de code JavaScript ainsi que les tests d'intégrations dans un projet logiciel. Vous allez vous familiariser avec des outils de tests tels que la librairie de tests Jest ainsi que les concepts de *Mock*, *Stub* et *Spy*.

Vous aurez à mettre en place les tests qui vérifient et valident le code de la logique du site web fait dans le TP2. Vous aurez également à implémenter une nouvelle fonctionnalité et à répondre à des questions de compréhension reliées à une problématique.

## 2 Introduction

Lors du deuxième travail pratique, vous avez mis en place la logique nécessaire pour rendre votre site web dynamique. Votre site web est maintenant capable de charger dynamiquement l'information nécessaire à afficher ainsi qu'à gérer l'ajout et la sauvegarde de nouvelles playlists à travers un formulaire modulaire.

Cependant, outre que les tests *Nightwatch* fournis, il n'y a aucune validation ou vérification de votre travail. Plus spécifiquement, le code JS que vous avez écrit n'est pas vérifié de manière directe. Un problème potentiel ne sera pas détecté que lorsque vous utilisez l'application.

## 3 Configuration du projet

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

Au cours de ce travail, vous aurez à utiliser les modules ECMAScript. La particularité de ces modules est qu'il n'est pas possible de les charger en utilisant le schéma URI `File`, c'est-à-dire, vous ne pouvez pas simplement ouvrir le fichier `index.html` à partir de votre ordinateur. Les ESM doivent être servis par un serveur HTTP.

Dans votre cas, vous utiliserez *lite-server*, un serveur statique qui vous permettra de faire un déploiement local d'un serveur HTTP qui fournit les fichiers sources 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.



## Avertissement

---

Afin de promouvoir la compréhension du langage JavaScript et la librairie, seulement la librairie de tests **Jest** est permise pour ce travail pratique. Vous utiliserez également la syntaxe ES2015 et les modules JS qui sont supportés par Jest.

---

## 3.2 Tests automatisés

Pour lancer la suite de tests de bout en bout (*e2e*), vous n'avez qu'à aller à la racine du répertoire `nightwatch` avec un terminal et de lancer la commande **`npm run e2e`**.

**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`** dans le répertoire `nightwatch`.

Les tests à compléter se trouvent dans le répertoire `tests/jest` du projet `siteWeb`. Notez que certains tests sont déjà faits pour vous afin de vous aider à démarrer le projet.

Pour lancer la suite des tests Jest, vous n'avez qu'à lancer la commande **`npm test`**. Vous pouvez également utiliser la commande **`npm run coverage`** qui lancera les tests et vous donnera un rapport de la couverture du code par la suite.

## 4 Travail à réaliser

À partir du code qui permet de répondre aux requis du travail pratique précédant, vous aurez à implémenter les tests nécessaires. Une version modifiée du TP2 vous sera fournie avec le dépôt de Git de départ afin de partir du bon pied pour ce travail. Notez que le comportement de cette version correspond au travail demandé au TP2, mais la structure interne n'est pas exactement pareille. Les tests de bout en bout de *Nightwatch* sont également placés dans un projet différent du site web afin de bien diviser les 2 parties du travail.

### 4.1 Éléments à réaliser

Maintenant que le site web est fonctionnel, la prochaine étape est de bien tester la logique d'implémentation. Avant de débiter, assurez-vous que le projet de départ est fonctionnel.

Vous aurez à implémenter une nouvelle fonctionnalité : Recherche par mots clés dans la bibliothèque. Vous mettrez en pratique les concepts de TDD (*Test Driven Development*) : les tests vous sont fournis et vous devez implémenter la logique correspondante.

## 4.2 Recherche dans la bibliothèque

La barre de recherche permet de récupérer les chansons et/ou playlists ayant un texte quelconque dans leur contenu : nom et description pour une *Playlist* et nom, artiste et genre pour une *Chanson*. Il est également possible de faire une recherche avec un *match* exact (sensible à la case) de mot clé.

Par exemple : la recherche suivante : **Co** affichera les 3 éléments suivants :

- *Playlist de Rock* (description : *Playlist avec une description beaucoup plus longue*)
- *Discover Weekly* (nom : *Discover Weekly*)
- La chanson *Bounce* (artist : *Coma-Media*)

Référez-vous aux figures 1 et 2 dans l'Annexe pour le résultat attendu.

Notez que la recherche est indépendante pour les chansons et les playlists. Il se peut que l'élément recherché se trouve seulement dans les playlists, seulement dans les chansons ou une combinaison des deux. Par exemple, la recherche **Playlist** devrait retourner que des playlists.

Une recherche vide devrait retourner tous les éléments normalement affichés dans la page. L'option de *match* exact ne devrait pas changer ce comportement.

Vous devez compléter les fonctions `searchInFields` et `search` dans `library.js` afin de répondre aux requis de la fonctionnalité. Une suite de tests unitaires dans le fichier `library.test.js` vous est fournie pour vous permettre de bien implémenter votre fonctionnalité. Assurez-vous de bien lire les tests unitaires pour vous aider à compléter la fonction.

Deux tests d'acceptation vous sont également fournis dans le fichier `index.js` dans le projet `nightwatch` (lignes 92 à 120). Ces tests vérifient la fonctionnalité en reprenant l'exemple présenté plus haut avec la recherche **Co** avec et sans *match* exact.

## 4.3 Vérification du code source

Le but principal de ce travail pratique est de mettre en place des tests unitaires pour le code qui vous est fourni. Vous trouverez les fichiers de test dans le répertoire `test/jest` déjà créés pour vous. Notez que certains tests unitaires sont déjà faits pour vous afin de vous aider à en écrire les autres. Vous avez également le titre des tests à compléter et qui vous donnera une idée du code à écrire pour compléter le test.

Assurez-vous de gérer les dépendances entre les différentes classes et/ou fonctions implémentées dans un test. Vous aurez à modifier le code fourni pour construire des `Stub`, `Mock` ou `Spy` lorsque nécessaire.



## Avertissement

---

Les tests à compléter sont vides, mais sont quand même considérés comme réussis lors de l'exécution de tous les tests. Ne vous fiez pas seulement à la sortie dans la console pour savoir si vous avez terminé votre travail. On vous recommande de placer la ligne suivante dans vos tests avant leur complétion pour s'assurer qu'un test non-terminé échoue : `expect(false).toBe(true);`

---

Les sections suivantes décrivent les différents fichiers de test à compléter. Notez que dans certains fichiers, vous n'avez qu'à compléter des tests et dans d'autres, vous devez implémenter quelques fonctions qui vous permettront de bien configurer votre échafaudage de tests.

Vous pouvez ajouter d'autres tests unitaires si vous le jugez nécessaire.

Dans le cas des fichiers `library.js` et `playlist.js`, le code dans les fonctions `onload` ne peut pas être couvert par des tests unitaires. Ceci est une limitation de la manière dont les événements sont gérés dans une page et nous acceptons que ces lignes ne soient pas couvertes.

### 4.3.1 `consts.test.js` et `utils.test.js`

Les tests dans ces fichiers vous sont fournis à titre d'exemple pour vous aider pour vos autres tests. Vous n'avez pas à tester les fonctions utilitaires qui vous sont fournis.

### 4.3.2 `library.test.js`

Ce fichier comporte 2 suites de tests. La deuxième suite vous est fournie pour vous aider à implémenter la fonctionnalité de recherche tel qu'expliqué dans la section 4.2. Vous pouvez utiliser ces tests pour vous aider avec le reste de votre travail.

Vous devez compléter la fonction `setUpHTML()` qui configure un DOM minimal pour vos tests. Vous pouvez vous inspirer du code déjà présent qui contient un exemple d'échafaudage pour la barre de recherche. Vous devez compléter la configuration pour les éléments conteneurs pour les playlists et les chansons. Référez-vous au code dans `index.html` et `library.js`.

ok

Vous devez compléter les tests unitaires marqués avec un **TODO** dans ce fichier. Le test *load should load window* vous est donné à titre d'exemple pour la bonne utilisation des *Spy*.

### 4.3.3 `player.test.js`

La configuration des tests dans ce fichier vous est fournie à titre d'exemple. Vous pouvez vous en inspirer pour vous aider à compléter les autres tests.

Certains tests vous sont déjà fournis à titre d'exemple. Vous pouvez les utiliser pour vous aider avec le reste des tests.

Vous devez compléter les tests unitaires marqués avec un **TODO** dans ce fichier.

ok

#### 4.3.4 playlist\_editor.test.js

Vous devez compléter la fonction `setUpHTML()` qui configure un DOM minimal pour vos tests. Vous pouvez vous inspirer du code déjà présent qui contient un exemple d'échafaudage pour certains éléments. Vous devez compléter la configuration pour les éléments de `DataList` et le bouton pour l'ajout d'une chanson. Référez-vous au code dans `create_playlist.html` et `playlist_editor.js`.

Vous devez compléter les tests unitaires marqués avec un **TODO** dans ce fichier. Les tests pour le chargement des images vous sont donnés déjà.

ok

#### 4.3.5 playlist.test.js

Vous devez compléter la fonction `setUpHTML()` qui configure un DOM minimal pour vos tests. Vous pouvez vous inspirer du code déjà présent qui contient un exemple d'échafaudage pour certains éléments. Vous devez compléter la configuration pour les éléments de la barre de progression ainsi que le reste des boutons. Référez-vous au code dans `playlist.html` et `playlist.js`.

ok

Vous devez compléter les tests unitaires marqués avec un **TODO** dans ce fichier. Les tests pour la gestion des raccourcis vous sont donnés déjà.

#### 4.3.6 storageManager.test.js

La configuration dans ce fichier vous est fournie à titre d'exemple. Vous devez compléter les tests unitaires marqués avec un **TODO** dans ce fichier. Certains tests vous sont donnés déjà pour vous aider avec votre travail.

ok

### 4.4 Questions de compréhension

En tant qu'équipe de développeurs, vous êtes confrontés à la problématique de conception suivante en lien avec votre projet en cours :

Vous aimeriez rendre votre recherche par mot clé dynamique, c'est-à-dire, effectuer la recherche et voir les résultats mis à jour pendant que vous tapez chaque lettre, et non à travers un clic de bouton à la fin (similaire à ce que Spotify fait sur leur site web).

Répondez aux questions suivantes dans un fichier nommé **questions.txt** (à remettre dans votre entrepôt Git avec la remise) :

1. Quels seraient les changements nécessaires afin d'implémenter cette fonctionnalité ?
2. Est-ce qu'il y a des problèmes potentiels avec cette approche ? **Justifiez.**
3. Quels tests devriez vous mettre en place pour bien tester cette fonctionnalité ? Ne vous limitez pas aux tests unitaires : considérez tous les types de tests vus dans le cours.

## **i** Conseils pour la réalisation du travail pratique

---

1. Séparez bien les classes testées de leur dépendances. Faites un usage des concepts de *Stub*, *Mock* et *Spy* de Jest.
  2. Lisez bien les tests fournis pour vous aider à implémenter la fonctionnalité manquante et les tests unitaires à compléter.
  3. Consultez [l'exemple de test unitaires avec Jest vu dans le cours](#).
  4. Exécutez les tests fournis souvent afin de valider votre code.
  5. Ajoutez des tests si vous jugez nécessaire.
  6. Respectez la convention de codage établie par *ESLint*. Utilisez la commande `npm run lint` pour valider cet aspect.
- 

## 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 : **tp3\_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 **14 mars**.

**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. Plus précisément, le barème de correction est le suivant :

Exigences	Points
Recherche dans la bibliothèque	
Respect des exigences fonctionnelles de l'énoncé	3
Tests unitaires	
Tests dans <code>library.test.js</code>	2
Tests dans <code>player.test.js</code>	3
Tests dans <code>playlist_editor.test.js</code>	3
Tests dans <code>playlist.test.js</code>	4
Tests dans <code>storageManager.js</code>	2
Qualité du code	
Qualité et clarté du code	1
Questions de compréhension	
Réponses dans un fichier <b>questions.txt</b>	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.

L'évaluations des tests se fera à travers le projet `nightwatch` ainsi que les tests dans le projet `siteWeb`. Tous les tests fournis doivent obligatoirement passer. Tous les tests unitaires écrits par vous doivent obligatoirement passer.

Le code doit respecter les règles établies par *ESLint*. La commande *lint* ne doit pas indiquer des erreurs dans la console après son exécution.


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



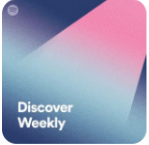
# Annexe

☐ Exact

**Mes Playlists**



Playlist de Rock  
Playlist avec une descr...



Discover Weekly  
Playlist avec beaucoup ...

**Mes Chansons**

Bounce

Electronic

Coma-Media

FIGURE 1 – Recherche de chaîne **Co** sans match exact

☒ Exact

**Mes Playlists**

**Mes Chansons**

Bounce

Electronic

Coma-Media

FIGURE 2 – Recherche de chaîne **Co** avec match exact