1 SPA monolithique & JSON

Veuillez créer une application web, une SPA sous Express, permettant d'enregistrer ,de manière persistante, des films et d'afficher les films enregistrés que pour les utilisateurs authentifiés.

Vous allez intégrer l'enregistrement et l'authentification d'un utilisateur fournis dans le dossier /demo/spa/spa-monolith du repo associé au cours.

L'exercice « 2 MPA, MVC et les sessions côté serveur» de la fiche 06 permet aussi d'enregistrer et visualiser des films, mais l'application rend son contenu HTML au niveau du serveur (Server Side Rendering).

Dans cet exercice, l'application doit rendre son contenu HTML dans le browser grâce à du JS (Client Side Rendering) et des appels à une REST API.

Tout comme précédemment :

- Votre application contiendra un menu avec deux liens : **Add film**, **List films**.
- Un utilisateur non authentifié n'a pas accès à ces deux liens.
- Par défaut, une fois un utilisateur authentifié, votre application doit afficher le contenu associé au lien **List films**.
- Un clic sur le lien **Add film** permet l'ajout d'un film à l'aide d'un formulaire.
  Votre formulaire doit contenir les champs :
  - **title** : titre du film
  - **duration** : durée du film en minutes
  - **budget** : pour informer du coût qu'a couté la production du film, en millions
  - **link** : pour donner un lien vers la description du film (lien vers imdb, rottentomatoes ou autre)
- Au submit valide du formulaire, vous allez rediriger l'application vers l'affichage de la liste des films. Cette affichage est aussi accessible lors d'un clic sur le lien **List films**.
  Voici un exemple d'affichage :

Title	Duration (min)	Budget (million)
Harry Potter and the Philosopher's Stone	152	125
Avengers: Endgame	181	181

…		

- Veuillez noter que le champs **link** est utilisé pour créer un lien dans la colonne **Title**.

Afin de réaliser cet exercice, voici les contraintes d'implémentation :

- Veuillez créer une RESTful API pour la collection de films, qui répond sur le port 800 et qui couvre ces fonctions :

URL	Méthode	Fonction
films	GET	READ : Lire toutes les films de la collection
films/1	GET	READ : Lire le film dont l'id vaut 1
films	POST	CREATE : Créer (enregistrer) un film basée sur les données de la requête
films/9	DELETE	DELETE : Effacer le film dont l'id vaut 9
films/9	PUT	UPDATE : Remplacer l'entièreté de la ressource par les données de la requête

- Les données doivent être persistantes : dès lors, sauvez les données associées aux films dans un fichier **.json**
- Assurez-vous que votre fichier de données **.json** n'est pas tracké : celui-ci ne doit pas être repris dans le web repository.
- Veuillez tester toutes les fonctions de la RESTful API pour la collection de **films** à l'aide du **REST Client** dans VS Code (extension à installer au sein de VS Code). Veuillez ajouter vos requêtes au sein du fichier **film-requests.http** dans le répertoire **/restClient** du dossier associé à cet exercice.
- Via votre SPA, veuillez consommer ces ressources à l'aide la méthode **fetch()**, sans utiliser **await** / **async** :
    - **GET films** pour la fonction d'affichage des films
    - **POST films** pour l'ajout d'un film via un formulaire
- Pour rappel, votre SPA ne charge ses fichiers statiques (**index.html** & Co) que lors du 1er appel au serveur, ou lors d'un refresh. L'affichage des composants de la page se fait via un routeur de composants par le biais de votre browser.

Une fois votre SPA fonctionnelle, veuillez tester ce qui se passerait si l'application ne renvoyait pas toujours **index.html** lors d'une demande de chargement d'une ressource statique.

Pour ce faire, vous pouvez commentez ce code au sein de app.js et réalisez des refresh de votre application quand, par example, vous êtes sur la page d'affichage des films.

```
app.use((req, res, next) => {
  if (!req.path.startsWith("/api/"))
    return res.sendFile(`${__dirname}/public/index.html`);
  next();
});
```

Challenge optionnel N°1 : si vous vous sentez l'âme d'un pirate, où d'un agent de sécurité, que vous avez fini tous les exercices de la semaine, n'hésitez pas à vous attaquer à ce challenge optionnel. Tentez d'injecter du javascript à l'aide du formulaire d'ajout de films. Cette injection doit faire en sorte que dès que vous affichez la liste des films, un pop-up d'alerte affiche un message.

Challenge optionnel N°2 : vous vous êtes pris au jeu ? Hé bien tentez de mettre à jour votre SPA pour qu'elle ne soit plus sujette à ce genre d'injection de Javascript.

- *Pour partir d'une application offrant l'enregistrement et l'authentification d'un utilisateur, vous pouvez démarrer votre développement en copiant le contenu de la démo /demo/spa/spa-monolith dans votre dossier associé à cet exercice.*
- *Pour créer le router de votre API au sein de **/routes/films.js**, vous devriez vous inspirer du code donné dans la même démo (/demo/spa/spa-monolith)*
- *Pour ne pas tracker un fichier au sein du repository local, et donc au sein du web repository aussi, ajouter son nom au sein de **.gitignore***
- *Si vous utilisez **nodemon** pour démarrer votre application, à chaque ajout de donnée dans un fichier .json, votre serveur va redémarrer. Vous allez donc perdre votre session, ce qui rend le test de votre RESTful API plus long. N'hésitez pas à mettre vos fichiers **.json** dans le répertoire **/data**. Pour que **nodemon** ignore les mises à jour dans ce dossier, vous pouvez ajouter cela à **package.json** :*

```
"nodemonConfig": {
    "ignore": ["data/*"]
  },
```

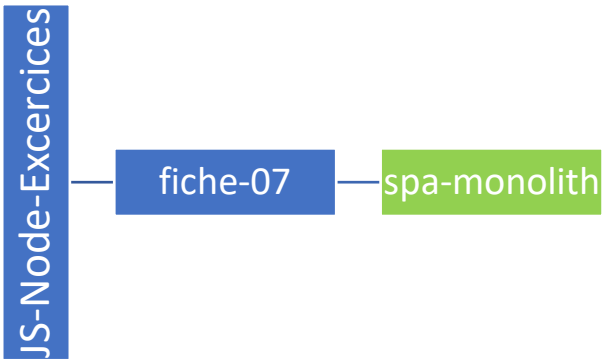
Exercices

- *Pour tester vos RESTful API associées à une gestion de session côté serveur, vous devez fournir un cookie à l'API. Pour ce faire, nous vous recommandons de créer un utilisateur au sein de votre application. Lorsque vous faite une requête, à l'aide de **REST Client**, vers une RESTful API qui renvoie un cookie (par exemple une requête de type POST http://localhost:800/api/users/login), REST Client sauvegarde ce cookie de manière transparente pour être envoyé lors de vos futures requêtes. Cela vous permettra de tester toutes les opérations sur les ressources qui demandent une autorisation. Voici un exemple de 3 requêtes pour pouvoir ajouter un film :*

```
### 0. Create the user james
POST http://localhost:800/api/users/
Content-Type: application/json

{
    "email":"james@cool.com",
    "password":"Jj"
}

### 1.Log the user james (the tool remember the cookies for subseque
nt requests !)
POST http://localhost:800/api/users/login
Content-Type: application/json

{
    "email":"james@cool.com",
    "password":"Jj"
}

### 2. Add a film (the session cookie linked to the login of james
will be automatically sent)
POST http://localhost:800/api/films/
Content-Type: application/json

{
    "title":"Star Wars: Episode 2",
    "duration": "1",
    "budget": "11",
    "link":"findIt.com"
}
```

Exercices

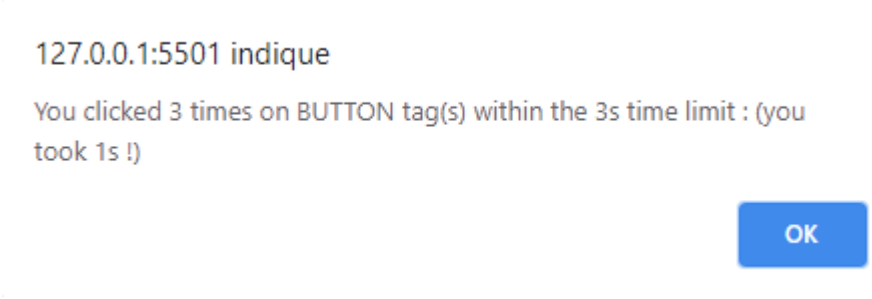
Le code de votre application doit se retrouver dans ce dossier (en vert) de votre repository local et de votre web repository (**JS-Node-Exercices**).



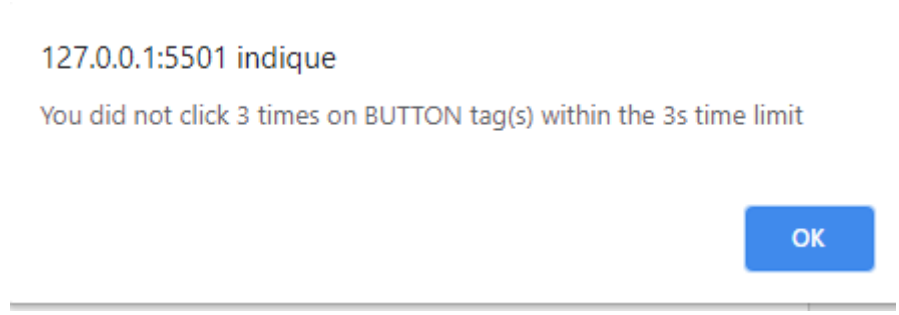
2 Exercice optionnel : la création de promesses

Vous pourriez adapter l'application web de gestion des clics sur des boutons fournie dans le dossier /demo/dom-event/click-button du repo associé au cours.

Cette nouvelle application afficherait un message d'alerte après qu'un utilisateur ait cliqué un nombre de fois paramétrable sur des boutons. Ce message d'alerte fournirait le temps, en seconde, entre le 1er clic et le dernier clic.
Par exemple :



S'il n'a pas cliqué assez rapidement, un autre message d'alerte serait affiché automatiquement à la fin du délai donné (**timeLimit** décrit ci-dessous).
Par exemple :



Afin de réaliser cet exercice, voici les contraintes d'implémentation :

- Veuillez créer une fonction qui renvoie une promesse, s'appelant : **clickMonitor(clicNumberThreshold, timeLimit)**
- La promesse résout si l'utilisateur a cliqué le nombre de fois correspondant à l'argument **clicNumberThreshold** sur des boutons.
- Elle est rejeté si l'utilisateur n'a pas cliqué endéans le nombre de secondes correspondant à **timeLimit**
- Veuillez appeler votre fonction **clickMonitor()** au chargement de l'application. Le retour de cette fonction fournit les données pour le message d'alerte.
  Tips : utilisez **.then()** et **.catch()** pour créer vos pop-ups d'alerte.
- **Challenge optionnel N°1** : Pour les plus avancés, en optionnel, n'hésitez pas à modifier votre code pour appeler votre fonction **clickMonitor()** à l'aide de

Challenge optionnel N°2 : Modifiez la fonction **clickMonitor()** pour qu'elle permette de monitorer les clicks seulement sur le type de tags indiqué dans un 3ème paramètre de votre code (monitoring de clicks que sur les td, ou les buttons ou le body…).

Le code de votre application doit se retrouver dans ce dossier (en vert) de votre repository local et de votre web repository (**JS-Node-Exercices**).