ElectronJS : TP - Todo List

> Vous allez créer une application de gestion de médias en Electron !

> La correction complète peut se trouver sur le lien suivant : https://github.com/tomsihap/hb-electron-tp2 . Pour l'installer, téléchargez le projet, ouvrez un terminal dans le projet et saisissez `npm install`. Vous pourrez ensuite le lancer avec `npm start`.

Installation

1. Créez un nouveau dossier nommé "mediatheque"
2. Créez un projet Electron dedans (créez les fichiers de base nécessaires, remplissez le `package.json`, installez Electron...)

Mise en place de l'interface

1. Créer la fenêtre principale de taille 800*600 qui contiendra le fichier `index.html`.
2. Composer le fichier `index.html` avec du HTML/CSS basique le template suivant :

```
            Titre du projet
            ---------------


 ____________________________________________________________
| [type de média] Titre du média 1  - [Bouton: "supprimer"] |
 ------------------------------------------------------------

 ____________________________________________________________
| [type du média] Titre du média 2  - [Bouton: "supprimer"] |
```

```
-------------------------------------------------------------

_____________________________________________________________
| [type du média] Titre du média 3  - [Bouton: "supprimer"] |
-------------------------------------------------------------

[Bouton: "Ajouter un élément"]
```

> Type de média: livre, musique, film, jeu...

3. Ajouter un évènement au bouton "Ajouter un élément" qui ouvre une nouvelle fenêtre de formulaire d'ajout d'élément :
    - Créer le fichier gérant la "logique" de cette fenêtre : `index.js`
    - Importer `index.js` dans `index.html`
    - Dans `index.js` : ajouter un event listener au bouton
    - Dans la fonction fléchée de l'event listener : créer un nouveau `BrowserWindow` qu'on nommera `winAddElement`.
    - Cette fenêtre contiendra le fichier `add-element.html`
    - Dans `add-element.html`, ajoutez les champs suivants :
        - input (texte) pour le nom de l'élément
        - select/option pour le type de l'élément
        - un bouton de validation
4. Quand on clique sur le bouton de validation du formulaire, afficher dans le console.log les données de l'input et du select
    - Ajouter un event listener sur le bouton de validation
    - Trouver un moyen de récuperer la `value` de l'input et du select
    - Dans la fonction fléchée de l'event listener, les afficher dans un `console.log`

```php
<?php

function hello() {
    return "hello !";
}
```

Traitement des données du formulaire

Maintenant que les données du formulaire sont bien reçues, nous allons faire en sorte de les enregistrer dans un espace de stockage.

En général en Javascript, on peut stocker les données avec LocalStorage :

```
// Enregistrer une donnée :
localStorage.setItem('username',  'John Doe')

// Récupérer cette donnée ailleurs dans l'application :
```

```
localStorage.getItem('username')
```

Les données sont disponibles globalement dans l'ensemble de l'application.

On veut créer et stocker un tableau d'objets contenant nos éléments en localStorage, par exemple :

```
[
    {
        title: '1917',
        type: 'movie'
    },
    {
        title: '1Q84',
        type: 'book'
    },
]
```

Le localStorage ne peut contenir que des *strings*, si on veut enregistrer un tableau, nous devons donc le convertir en string avant de l'enregistrer :

```
let arrayElements = [
    {
        id: 1,
        title: '1917',
        type: 'movie'
    },
    {
        id: 2,
        title: '1Q84',
        type: 'book'
    },
];

localStorage.setItem('elements', JSON.stringify(arrayElements))
```

Et si on veut le récupérer, on doit décoder la string enregistrée en un tableau Javascript :

```
let arrayElements = JSON.parse(localStorage.getItem('elements'))
```

Exercice : stocker les données enregistrées dans localStorage

1. Vérifier si des éléments existent dans le localStorage pour la clé `elements`. Si oui, les récupérer, *parsés*, dans une variable intérmédiaire `elementsArray`. Si non, créez un nouveau tableau `elementsArray`.

2. Ajoutez le nouvel élément créé dans le tableau `elementsArray`. L'élément créé sera sous le format suivant:

```
{
    titre:  "Titre de l'élément",
    type: "Type de l'élément
}
```

3. Stocker le tableau `elementsArray`, *stringifié* dans la clé `elements` du localStorage.

Indices

Voici un exemple du code que vous pourriez avoir :

```
const formInput = document.getElementById('form-input')
const formType = document.getElementById('form-type')
const formSend = document.getElementById('form-send')

formSend.addEventListener('click', function() {

    alert('Vous avez créé le ' + formType.value + ' nommé ' +
formInput.value )

    // Créez la variable "element" qui contient un objet, avec les données
de l'utilisateur

    let element = {
        ...A_REMPLIR...
    }

    // Créez le tableau "elementsArray", vide, pour avoir par défaut un
tableau
    // d'éléments vide que l'on déclare
    let ...A_REMPLIR...

    // Vérifiez si la clé "elements" existe dans le localStorage. Pour
cela :
    if ( tableau_element_dans_local_storage !== null  ) {

        // Si le tableau "elements" existe dans localStorage, alors on
l'assigne à elementsArray
        // pour récupérer les éléments existants
        elementsArray =
tableau_element_dans_local_storage_en_version_JSON.parse
    }

    // Ajoutez l'élément utilisateur (variable "element") au tableau
"elementsArray"
    // grâce à push() : tableau.push(element_a_rajouter)
```

```
    ...A_REMPLIR...

    // Enfin, enregistrez en version strigifiée le  tableau elementsArray
dans la clé "elements" du localStorage

    ...A_REMPLIR...

})
```

Correction

Voici un exemple du code à produire :

```
const formInput = document.getElementById('form-input')
const formType = document.getElementById('form-type')
const formSend = document.getElementById('form-send')


formSend.addEventListener('click', function() {

    alert('Vous avez créé le ' + formType.value + ' nommé ' +
formInput.value )

    // Créez la variable "element" qui contient un objet, avec les données
de l'utilisateur

    let element = {
        title: formInput.value,
        type: formType.value
    }

    // Créez le tableau "elementsArray", vide, pour avoir par défaut un
tableau
    // d'éléments vide que l'on déclare
    let elementsArray = [];

    // Vérifiez si la clé "elements" existe dans le localStorage. Pour
cela :
    if ( localStorage.getItem('elements') !== null  ) {

        // Si le tableau "elements" existe dans localStorage, alors on
l'assigne à elementsArray
        // pour récupérer les éléments existants
        elementsArray = JSON.parse(localStorage.getItem('elements'))
    }

    // Ajoutez l'élément utilisateur (variable "element") au tableau
"elementsArray"
    // grâce à push() : tableau.push(element_a_rajouter)
    elementsArray.push(element)
```

```
    // Enfin, enregistrez en version strigifiée

    localStorage.setItem("elements", JSON.stringify(elementsArray));

})
```

Communiquer à une autre fenêtre la mise à jour du localStorage: fenêtres parents et enfants

> Maintenant que notre tableau contient des données, nous devons indiquer à la fenêtre contenant `index.html` qu'une donnée a été ajoutée au `elements` de localStorage de sorte à ce que la fenêtre contenant `index.html` mette à jour l'affichage.

Nous allons faire en sorte de communiquer de fenêtre en fenêtre (depuis la fenêtre `add-element` vers la fenêtre `index`). Nous allons utiliser la notion de fenêtre enfant et fenêtre parent : en effet, de fenêtre enfant à parent, ou de parent à enfant, on peut envoyer des informations avec Electron !

Pour cela, lorsque vous déclarez une nouvelle fenêtre, il faut la déclarer en fenêtre enfant :

```
// En haut du fichier, ajoutez cette ligne pour importer le module
"remote"
const remote = require('electron').remote;


// À la  déclaration d'une fenêtre, ajoutez un attribut pour déclarer le
parent :
let win = new BrowserWindow({
    // ...,
    parent: remote.getCurrentWindow(),
})
```

- `remote` nous permet d'utiliser les outils dédiés aux process de rendus (les fenêtres)
- `getCurrentWindow()` réfère à la fenêtre dans laquelle on est en train d'exécuter ce code. La fenêtre parente !
- `parent` on indique le parent de la fenêtre que l'on est en train de créer

On indique donc, à la fenêtre que nous créons, que son parent est la fenêtre dans laquelle on se trouve actuellement. Par exemple, `index` est le parent de `add-element`.

Exercice : ajouter un parent à `add-element`

- Modifiez le code dans `index.js` de sorte à ajouter un parent à la fenêtre qui ouvre la page `add-element.html`

Communiquer de fenêtre enfant à fenêtre parent

Maintenant que nous avons déclaré notre fenêtre enfant `add-element` comme ayant pour parent `index`, nous pouvons communiquer de l'enfant au parent !

Voilà comment faire.

Dans la fenêtre enfant (envoi du message/évènement)

1. Ajouter en haut du fichier le code suivant :

> Il s'agit des outils propres aux process de rendus

```
const remote = require('electron').remote;
```

2. Envoyer un message (un évènement), par exemple après un déclanchement d'un click, grâce à :

> On prend la fenêtre actuelle (`remote.getCurrentWindow()`), à laquelle on prend la fenêtre parent (`.getParentWindow()`), à laquelle on envoie un message de notre choix (`.send('message')`)

```
remote.getCurrentWindow().getParentWindow().send('titre-du-message-
declanche');
```

Dans la fenêtre parent (réception du message/évènement)

1. Ajouter en haut du fichier le code suivant :

> Il s'agit des outils de communication IPC (Inter Process Communication) qui permettront au parent de lire les messages reçus

```
const { ipcRenderer } = require('electron');
```

2. Ajouter dans le fichier, n'importe où dans le scope global, le code suivant :

> C'est un évènement écouté grâce à `on('message')` ! En réaction, on a une fonction anonyme (`function() {}`) dans laquelle on met le code que l'on veut qui sera déclanché à la réception du message.

```
ipcRenderer.on('titre-du-message-declanche', function() {
    console.log('Message reçu !')
})
```

Exercice : envoyer un message de parent à enfant

1. Créez un bouton "Envoi d'un message au parent" dans `add-element`.
2. Au clic sur ce bouton, envoyer le message `clicked-button` au parent

3. Dans le parent, à la réception du message `clicked-button`, affichez un console.log `Le bouton a été cliqué dans la fenêtre enfant`

> Note: attention, comme vous affichez le console.log dans le parent, ouvrez bien les Developer Tools dans le parent !

Correction

Dans `add-element.html` :

```
<button id="btn-send-message" class="btn btn-primary">Envoyer un message
au parent</button>
```

Dans `add-element.js` :

```
// En import en haut du fichier :
const remote = require('electron').remote;

// ...

// Plus loin dans le fichier :
const btnSendMessage = document.getElementById('btn-send-message');

btnSendMessage.addEventListener('click', function() {
    remote.getCurrentWindow().getParentWindow().send('clicked-button');
})
```

Dans `index.js` :

```
// En import en haut du fichier :
const { ipcRenderer } = require('electron');

// ...

// Plus loin dans le fichier :
ipcRenderer.on('clicked-button', function() {
    console.log('Le bouton a été cliqué depuis la fenêtre enfant !');
})
```

Exercice : Envoyer un message quand on créée un élément

Sur le même principe que l'exercice précédent :

- Depuis l'enfant, envoyez le message `element-added` au parent lorsqu'un nouvel élément a été ajouté au localStorage
- Depuis le parent : quand le message est reçu, faites un console.log "Un nouvel élément a été envoyé".

Correction

Dans `add-element.js` :

```
formSend.addEventListener('click', function() {

    // Dans l'eventListener qui écoute le clic sur le bouton d'ajout d'un
élément...
    //...
    localStorage.setItem("elements", JSON.stringify(elementsArray));
    // Après avoir enregistré le tableau dans localStorage...

    remote.getCurrentWindow().getParentWindow().send('element-added');
})
```

Dans `index.js` :

```
// à la fin du fichier :
ipcRenderer.on('element-added', function () {
    console.log('Un élément a été ajouté au localStorage !')
});
```

Afficher les données !

Ça y est ! On a réussi à :

- Ouvrir une fenêtre d'ajout d'un élément
- Ajouter un élément et l'enregister dans une base de données locale, localStorage
- Faire savoir à la fenêtre principale, `index`, qu'un élément a été ajouté

Il nous manque une dernière étape : maintenant que `index` sait quand un élément a été ajouté au localStorage, il faut qu'il lise ces élements depuis le localStorage pour les afficher dans le HTML.

Exercice

Dans `index.js`, dans la fonction anonyme qui réagit à l'évènement `element-added` de l'exercice précédent:

1. Récupérez le tableau d'éléments (parsé) qui contient tous les éléments ajoutés, issu du localStorage.
2. Faites une boucle sur ce tableau de sorte à console.log chaque élément du tableau de la façon suivante :
    - `console.log(title + ' ' + type)`
3. Plutôt que de les afficher en console.log, affichez-les dans le HTML.

Correction

Dans `index.js` :

```
ipcRenderer.on('element-added', function () {
    console.log('Un élément a été ajouté au localStorage !')

    // 1. On récupère les données depuis le localStorage, et on les parse
    let elementsArray = JSON.parse(localStorage.getItem('elements'))

    // 2. On fait une boucle forEach qui permet de scanner le tableau
    // et faire un console.log de chaque élément
    elementsArray.forEach(function(element, index) {
        console.log(element.type + " " + element.title)
    })

    // 3. On gère l'affichage dans le HTML

    // On récupère la balise HTML qui contiendra mes éléments
    const elementsList = document.getElementById('elements-list')

    // D'abord, on vide l'affichage existant si jamais il y avaitdéjà une
liste affichée dans elementsList
    elementsList.innerHTML = '';

    // Ensuite, on fait une boucle sur tous les éléments
    elementsArray.forEach(function(element, index) {

        // On prépare le code HTML d'un élément de la liste
        let elementHtml = '<li><strong>'+ element.type +'</strong> - '+
element.title +'</li>';

        // Ensuite, on ajoute au sein du HTML de notre balise
elementsList, l'élément à rajouter :
        elementsList.innerHTML += elementHtml;
    })
});
```

Et voilà ! Vous pouvez dorénavant ajouter des éléments dans le formulaire, et la liste se mettra à jour automatiquement.

Par contre, quand on quitte et réouvre l'application, notre liste se vide au lieu d'afficher les éléments existants. Pour cela, il suffit d'ajouter dans `index.js` précisément le code nous permettant d'afficher la liste des éléments que nous venons de faire.

Il faut ajouter ce code assez haut dans le fichier, par exemple juste après tous les `const/require` :

```
let elementsArray = JSON.parse(localStorage.getItem('elements'));

const elementsList = document.getElementById('elements-list');
elementsList.innerHTML = '';

elementsArray.forEach(function (element, index) {
    let elementHtml = '<li><strong>' + element.type + '</strong> - ' +
```

```
element.title + '</li>';
    elementsList.innerHTML += elementHtml;
})
```