

Formattage du temps

Player

- Par défaut, les valeurs transmises par le tag audio sont en secondes
- Le but est de les afficher proprement, sous forme “MM:SS” (ex: 03:24)
- Vous pouvez utiliser le code suivant:

<https://github.com/lgavillet/webmobui-22/blob/master/Ressources/formatTimestamp.js>

Formattage du temps

Player

```
import formatTimestamp from './lib/formatTimestamp' // ou autre chemin que vous aurez choisi
```

```
...
```

```
console.log(formatTimestamp(duréeEnSecondes))
```

Recherche

Structure

Recherche



Comment intégrer la fonction de recherche ?

Structure

Recherche

- Créer un listener 'change' sur le champ de recherche
- Quand il change, il faut prendre sa valeur et la passer à l'endpoint de recherche
- La valeur se passe dans l'URL: /api/songs/search/:query
- Exemple: /api/songs/search/dynoro

Structure

Recherche

- Cela fonctionne bien pour de petites chaines, mais comment passer des caractères complexes ?
- `encodeURIComponent(laChaine)`
- Il faudrait donc faire:
- `fetch("https://api/songs/search/" + encodeURIComponent(laChaine))`

Structure

Recherche

- L'API de recherche vous renvoie la même structure que l'endpoint des chansons d'un artiste
- Vous pouvez donc réutilisez une bonne partie de votre code pour afficher résultat!
- Une fois le résultat obtenu, il vous suffira d'afficher la section correspondante pour la liste des résultats (de l'intérêt de bouger la logique d'affichage dans des petites méthodes d'aide :))

Playlists

Concept

Playlists

- Le but est de gérer les playlists localement
- En opposition à l'API qui gère les chansons de manière centralisée sur le serveur, nous utiliserons une des API Web pour les stocker dans le browser
- Plusieurs API disponibles...

Les API de stockage

Playlists

APIs de stockage :

- IndexedDB - Une vraie database, disponible dans le browser
- Web Storage - Stockage de clés/valeurs (genre de cookies sous stéroïdes)

Nous allons nous concentrer sur la deuxième...

L'API Web Storage

Playlists

Web Storage est séparée en 2 sous-API :

- **sessionStorage**
Les données ne sont stockées que pendant la durée de la session. Lorsque le browser se ferme, les data sont effacées
- **localStorage**
Les données sont persistées entre les sessions et sont présentes après un restart

Ces deux APIs s'utilisent de la même manière, c'est uniquement la persistance des données qui change

L'API Web Storage

Playlists

Méthodes principales :

- `setItem(cle, valeur)` - Définit un élément "cle" avec comme valeur "valeur"
- `getItem(cle)` - Lit l'élément "cle"
- `removeItem(cle)` - Supprime l'élément "cle"
- `clear()` - Vide la totalité du localStorage

L'API Web Storage

Playlists

Exemple :

```
const uneValeur = 12345  
  
localStorage.setItem( 'uneCléAChoix', uneValeur )  
  
console.log( localStorage.getItem( 'uneCléAChoix' ) )  
  
==> "12345"
```

L'API Web Storage

Playlists

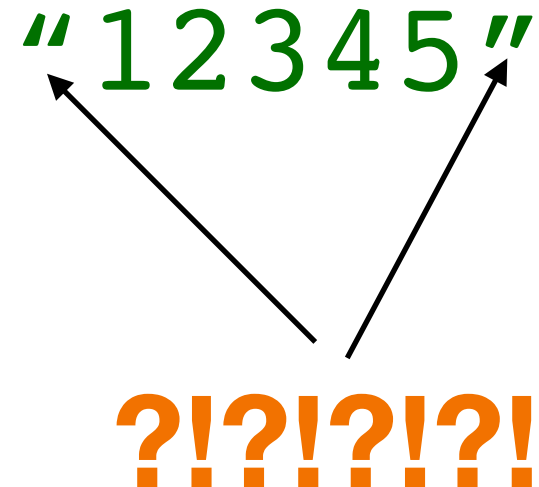
Exemple :

```
const uneValeur = 12345
```

```
localStorage.setItem( 'uneCléAChoix', uneValeur )
```

```
console.log( localStorage.getItem( 'uneCléAChoix' ) )
```

==> "12345"



?!?!?!?!

localStorage ne stock que des chaînes de caractères...

L'API Web Storage

Playlists

- Est-ce un problème ?
- Oui et non... nous connaissons... JSON !
- Il est donc possible de créer deux petites fonctions d'aide

```
function setJsonItem(clé, valeur) {  
    return localStorage.setItem(clé, JSON.stringify(valeur))  
}  
  
function getJsonItem(clé) {  
    return JSON.parse(localStorage.getItem(clé))  
}
```

L'API Web Storage

Playlists

- Cela fonctionne pour les structures standards, mais dès qu'il y a de grosses structures, cela devient complexe à gérer
- Il faudrait donc améliorer notre API custom pour gérer des cas plus complexe
- Exemple: Plusieurs playlists...
`setJsonItem('playlist_1', [...])`
`setJsonItem('playlist_2', [...])`

L'API Custom JsonStorage

Playlists

- M. Chabloz vous a préparé une API plus complexe qui permet de gérer facilement ces grosses structures

<https://github.com/lgavillet/webmobui-22/blob/master/Ressources/JsonStorage.js>

L'API Custom JsonStorage

Playlists

- Comment importer et instancier l'API JsonStorage :

```
import JsonStorage from './lib/jsonStorage' // ou autre chemin que vous aurez choisi
```

```
...
```

```
const playlistStorage = new JsonStorage({ name: 'playlists' })
```

L'API Custom JsonStorage

Playlists

Les méthodes importantes :

- `addItem(valeur)` - Pour ajouter un élément, comme un tableau (cela va aussi générer un id aléatoire comme clé)
- `setItem(id, valeur)` - Modifier un élément
- `removeItem(id)` - Supprimer un élément
- `forEach(function(valeur, id))` - Itérer sur les éléments

L'API Custom JsonStorage

Playlists

Utilisation de base et ajout d'une playlist

```
import JsonStorage from './lib/jsonStorage' // ou autre chemin que vous aurez choisi  
  
...  
  
const playlistStorage = new JsonStorage({ name: 'playlists'})  
  
...  
  
playlistStorage.addItem({ name: 'Ma playlist', songs: [...]})
```

L'API Custom JsonStorage

Playlists

Modifier une playlist

```
const maPlaylist = playlistStorage.getItem(idDeLaPlaylist)

maPlaylist.name = 'Un nouveau nom'

maPlaylist.songs = maPlaylist.songs.slice(1) // on enlève la première, par exemple..

playlistStorage.setItem(idDeLaPlaylist, maPlaylist)
```

L'API Custom JsonStorage

Playlists

Supprimer une playlist

...

```
playlistStorage.removeItem(idDeLaPlaylist)
```

L'API Custom JsonStorage

Playlists

Afficher les playlists

...

```
playlistStorage.forEach((valeur, id) => {  
    console.log("l'id est :", id)  
    console.log("le nom est :", valeur.name)  
    console.log("les chansons:", valeur.songs)  
    afficherPlaylist(valeur)  
})
```