1 Manipulations de fichiers

1.1 Données brutes : `Blob`

- un `Blob` (**B**inary **L**arge **Ob**jects) représente un objet qui comme un fichier contient des données brutes (pas dans un format javascript)
- Les données d'un `Blob` peuvent être lues comme texte ou comme données binaires, les méthodes suivantes renvoient une promesse tenue avec :
  - `text()` : une chaîne contenant le contenu du fichier interprété comme du texte UTF-8
  - `arrayBuffer()` : un tableau binaire `ArrayBuffer` contenant le contenu du fichier entant que données binaires
- Créer un Blob : constructeur `Blob(`*`tableauDonnées`*`[,`*`options`*`])`
  - `tableauDonnées` : tableau contenant un mélange de `ArrayBuffer`, `Blob` ou chaînes (en UTF-8)
  - `options` : objet avec propriété `type` (type MIME)

1.2 Données d'un fichier : `File`

- Un `File` donne des informations sur un fichier et permet d'accéder à son contenu
- Un `File` est un `Blob` : il a les méthodes de `Blob` et peut-être utilisé à la place d'un `Blob`
- Un `File` a les propriétés (en lecture) :
  - `name` : nom du fichier (sans la partie chemin)
  - `size` : taille en octets
  - `type` : type MIME
- Créer un `File` : constructeur `File(`*`tableauDonnées, nom, [,options`*`)`)
  - `tableauDonnées` : tableau contenant un mélange de `ArrayBuffer`, `Blob` ou chaînes (en UTF-8)
  - `options` : objet avec propriété `type` (type MIME)

1.3 Lecture d'un fichier local

- Javascript ne peut lire directement lui-même un fichier local
- Il faut demander un fichier à l'utilisateur pour pouvoir le lire avec un contrôle :
  - `<input type="file">` : sélectionner un seul fichier
  - `<input type="file" multiple>` : sélectionner plusieurs fichiers
- Qui émet un événement `change` à chaque changement de sélection
- L'élément du DOM correspondant a une propriété `files` :
  - pseudo-tableau de type `FileList` d'objets de type `File` représentant le(s) fichier(s) sélectionné(s)
  - Si un seul fichier sélectionné : c'est le premier du tableau

1.4 Accéder au contenu d'un Blob/fichier : `FileReader`

– On peut lire le contenu d'un `Blob` ou d'un `File f` de façon *asynchrone* avec un `FileReader`

1. Créer un `FileReader` : `const reader = new FileReader();`
2. Attacher des fonctions de rappel pour les événements `load`, `error` et `abort` :
   – `reader.onload = fnRappel;` ...
   – ou `reader.addEventListener('load',fnRappel)` ...
3. Lancer la lecture du `File f` en tant que :
   – `reader.readAsText(f[, encodage])` : texte suivant l'encodage donné (UTF-8 par défaut)
   – `reader.readAsArrayBuffer(f)` : données binaires
4. S'il faut avorter la lecture : `reader.abort()`

- S'il y a une erreur lors de la lecture : le `FileReader` émet l'événement `error`
- Si la lecture est avortée : le `FileReader` émet l'événement `abort`
- Quand la lecture s'achève avec succès :
  - le `FileReader` émet l'événement `load`
  - sa propriété `result` contient alors le contenu du fichier sous forme :
    - d'une chaîne si lu avec `readAsText`
    - d'un tableau binaire `ArrayBuffer` si lu avec `readAsArrayBuffer`
- Les fonctions de rappel sont appelées avec l'événement `e` en paramètre
- et `e.target` référence le `FileReader` qui a émis l'événement

1.5 Sauvegarde d'un blob/fichier

- Utiliser FileSaver : `github.com/eligrey/FileSaver.js`
  - charger `FileSaver.js` (`FileSaver.min.js` en production) dans la page HTML
  - ou importer `import { saveAs } from 'file-saver';` (`'file-saver.js'` si pas webpack)
- Sauver un fichier : `FileSaver.saveAs(f, nomFichier);` (`saveAs` si importé)
  - `f` : `Blob`, `File` ou URL
  - ne pas donner le nom du fichier si `f` est un `File` contenant déjà un nom
- Types MIME : texte : `text/plain`, JSON : `application/json`

2 Navigation

- `Window.location` (accessible en variable globale `location`) : objet de type `Location` représentant l'adresse du document
- Propriétés :
  - `href` : URL complète, si changée le document navigue vers la nouvelle page
  - parties de l'URL : `protocol`, `host`, `port`, `pathname`, `search`, `hash`
- Méthodes :
  - `assign(URL)` : charge la ressource à l'URL
  - `reload()` : recharge la ressource (`reload(true)` sans utiliser le cache)
  - `replace(URL)` : remplace la ressource par celle à l'URL (pas de trace dans historique du navigateur)

- `location="URL"` = `location.href="URL"` = `location.asign("URL")`
- Navigation interne : quand seule la partie `#...` du dièse de l'URL de l'adresse change,
  - le document ne change pas
  - le navigateur défile la page jusqu'à l'élément d'`id` correspondant
  - cet élément prend en plus la pseudo-classe CSS `:target`
  - `window` émet l'événement `hashchange` qui a pour propriétés
    - `oldURL` : l'ancienne URL de la page
    - `newURL` : l'URL vers laquelle on navigue