API de Navigateur

Une "API" est une "Application Programming Interface".

Ce sont des outils donnant accès à des fonctionnalités qui serait complexe et longue à développer soit même.

Il en existe deux types, les API de tierces que l'on verra dans une prochaine partie, et les api de navigateur.

Les API de navigateur sont des api inclu directement dans les navigateurs et généralement disponible sur n'importe quel navigateur moderne.

Voyons ensemble quelques exemples.

1. Fetch

Javascript est capable de lancer des requêtes HTTP à des fichiers sur votre serveur ou à des serveurs tiers.

Avant l'existence de l'API fetch, on devait écrire un code quelques peu complexe et faire appel à de nombreuses lignes de codes.

Avec l'API fetch, il devient possible de réduire cela à seulement quelques lignes de code peu nombreuse.

Exemple :

```
const url = "fichier.json"
// Avant Fetch :
const xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = handleRequest;
function handleRequest(){/* Gérer la requête*/}
xmlhttp.open("GET", url, true);
xmlhttp.send();
// Avec Fetch :
fetch(url).then(handleFetch);
function handleFetch(response){/* Gérer la requête*/}
```

2. fetch en GET

Lorsque l'on lance une requête **HTTP**, par exemple lorsque l'on charge une page. Différentes méthodes peuvent être utilisé, la principale et celle par défaut est **GET**.

De ce fait, histoire de simplifier les choses, l'utilisation de fetch avec get ne demande qu'à donner en paramètre de notre fonction fetch, le chemin du fichier ou l'url du site cible.

Pour faire passer des informations avec cette méthode, on affichera ces dernières directement dans l'url. C'est ce qui est souvent utilisé pour les moteurs de recherche.

Exemple :

```
fetch(url).then(handleFetch);
function handleFetch(response)
{
  if(!response.ok)return;
  response.text().then(function(data)
  {
    console.log(data);
  })
}
```

par exemple https://www.google.fr/search?q=pizza" recherchera "pizza" sur google.

3. fetch en POST

La méthode POST est la seconde plus utilisé, elle est utilisé afin de faire passer des informations de façon plus sécurisé entre le navigateur et le serveur cible.

Par exemple elle est utilisé à chaque fois que vous vous inscrivez ou vous connectez à un site.

Pour faire une requête avec cette méthode, Fetch va avoir besoin qu'on lui donne un second paramètre.

Celui ci contiendra un objet indiquant quel sera la méthode utilisé, et quel sera le corps (contenu) de la requête.

Exemple :

```
const options = {
  method: "POST",
  body: "information à envoyer"
  }
fetch(url, options).then(handleFetch);
function handleFetch(response)
{
  // Traitement de la requête
}
```

Voici une vidéo présentant les fonctionnalités de fetch si vous souhaitez pousser cela plus loins : https://www.youtube.com/watch?v=z9pcgJX1DdY

4. WebStorage

Une autre API de navigateur bien pratique est le **webStorage**. Il est divisé en deux objet distinct mais qui ont exactement les même propriétés et méthodes. Seul leur façon de sauvegarder des données varie.

Mais ne sauvegardons nous pas les données en Base de donnée ?
Certes pour des données sécurisé ou importante il vaut mieux sauvegarder en BDD. Mais pour des informations plus mineurs comme "Préférez vous votre site en thème sombre ou clair, en français ou anglais", et cela sans avoir à s'inscrire sur le site.
Alors on préfèrera sauvegarder les données directement sur le navigateur de l'utilisateur.

Est-ce cela que l'on nomme **cookie**?
Oui et non, au niveau réglementation l'utilisation du webstorage doit respecter les même règles que pour les cookies mais ce ne sont pas des cookies.
Les cookies sont sauvegardé sur le navigateur et échangé à chaque requête HTTP avec le serveur.
Le webstorage est sauvegardé uniquement sur le navigateur et ne bouge pas.

Les deux objets dont on parle avec webstorage sont :

- **localStorage**
- **sessionStorage**

La seule différence est donc leur façon de sauvegarder. **localStorage** va sauvegarder la donnée sur votre navigateur et elle n'y bougera pas tant qu'elle n'aura pas été supprimé manuellement ou par le site. **sessionStorage** supprimera la donnée dès que le navigateur sera fermé.

Selon l'utilité qu'on en a on utilisera plutôt l'un ou l'autre.

Exemple :

```
localstorage.setItem("color", "orange");
localstorage.getItem("color");
localstorage.removeItem("color");
```

Encore une fois, si vous souhaitez pousser cela plus loins, une vidéo explicative :
https://www.youtube.com/watch?v=ITmKqkmHlnY

5. canvas

Le canvas est à la base une balise HTML qui ne sert à rien. Plus sérieusement elle gagne sont utilité grâce à javascript.

Dans cette balise nous allons pouvoir grâce à l'API de canvas dessiner et animer du contenu.

On va commencer par lui indiquer un context (souvent "2D" mais si on la pousse plus loins il est possible de faire de la "3D" avec des outils célèbres pour cela).

Depuis ce context on va pouvoir lui indique des formes, des lignes, du texte ou des images à placer à tel ou tel endroit et avec une couleur choisi.

Avec les bonnes fonctions il est possible de redessiner nos éléments à rythme régulier en les déplaçant légèrement pour faire des animations.

Exemple :

```
const canvas = document.querySelector("canvas");
const context = canvas.getContext("2D");
context.fillStyle = "red";
context.fillRect(10, 20, 120, 50);
```

Voici un développeur de plus qui rentrera dans les détails : https://www.youtube.com/watch?v=FDBHLX5HFn0

6. three.js

J'ai dit plus tôt qu'il est possible de faire de la 3D dans les canvas. Mais cela demande des commandes complexe et qui ne sont pas à la portée de tous, des tas de calcul peuvent entrer en compte. Cela peut être assez dissuasif.

Mais ne fuyez pas, des gens ont fait le travail pour vous, par exemple la bibliothèque "**three.js**" permet de faire de la 3D de façon simplifier dans vos canvas.

Une fois que vous gérez bien l'API canvas, il vous suffira de lire la documentation de cette bibliothèque pour vous mettre à la 3D.

Pour plus d'exemple, voici une vidéo : https://www.youtube.com/watch?v=4IvhajhllFo

7. Activité

Mettons cela en pratique avec quelques exercices :

1. Requête à un fichier JSON

Créons un fichier **data.json** avec le contenu suivant :

```
{
  "fr": "Mon site est en français",
  "en": "Mon site est en anglais"
}
```

Créez dans votre HTML une liste à choix permettant de choisir entre "Français" et "Anglais", ainsi qu'un titre "h1".

Grâce aux écouteurs d'évènement "**input**" ou "**change**" faites une requête au fichier "**data.json**" via l'api **fetch**.

Une fois les données obtenu, changez le titre de la page selon la langue choisi par l'utilisateur.

2. Sauvegarder le choix de l'utilisateur

Une fois que l'utilisateur a choisi sa langue, sauvegardez le choix de l'utilisateur via l'API de **web storage**.

Ajoutez un bouton dans votre HTML et au **clique** sur celui ci, chargez le choix sauvegardé par l'utilisateur.

3. Canvas Adaptable

Ajouter une balise canvas à votre HTML.

Grâce à l'évènement **resize** faites que la taille de votre canvas change selon la taille de votre fenêtre.

Utilisez l'évènement **mousemove** pour afficher le titre dans le canvas avec la langue précédement choisi. Puis faites lui suivre le mouvement de la souris.