Partie 1 : JS avancé et outils de débogage

Dans cette première partie, nous allons nous concentrer sur les bases de JS pour construire une PWA au goût du jour (en date du milieu 2022). Pour ce cas nous allons voir :

1. Les fonctions fléchées
2. L’asynchronicité en JS
3. Différents outils de débogage

JS and the fabulous arrow function

old one

```
let x = myFunction(4, 3);
function myFunction(a, b) {
  return a * b;
}
console.log(x); //output = 12
```

new one

```
const myfunction = (a, b) => a * b;
console.log(myfunction(4, 3)); //output = 12
```

Less is more

Les fonctions fléchées surtout appelées "arrow function" ont été introduites dans JS pour compacter le langage et le rendre plus lisible et moins onéreux en lignes de code.

Particularités

- Dès que la fonction doit s'étaler sur plusieurs lignes (pour une meilleure lisibilité par exemple) les accolades *{ }* doivent être présentes et le mot réservé *return* doit être rajouté.

```
const myfunction = (a, b) => {
  let x = a * b;
  if (x > 3) return "hello world";
};
console.log(myfunction(4, 3)); //output = hello world
```

- 2 points importants : les arrow function ne peuvent être utilisées pour un constructeur (orienté objet) ni en tant que méthode dans une classe par exemple.

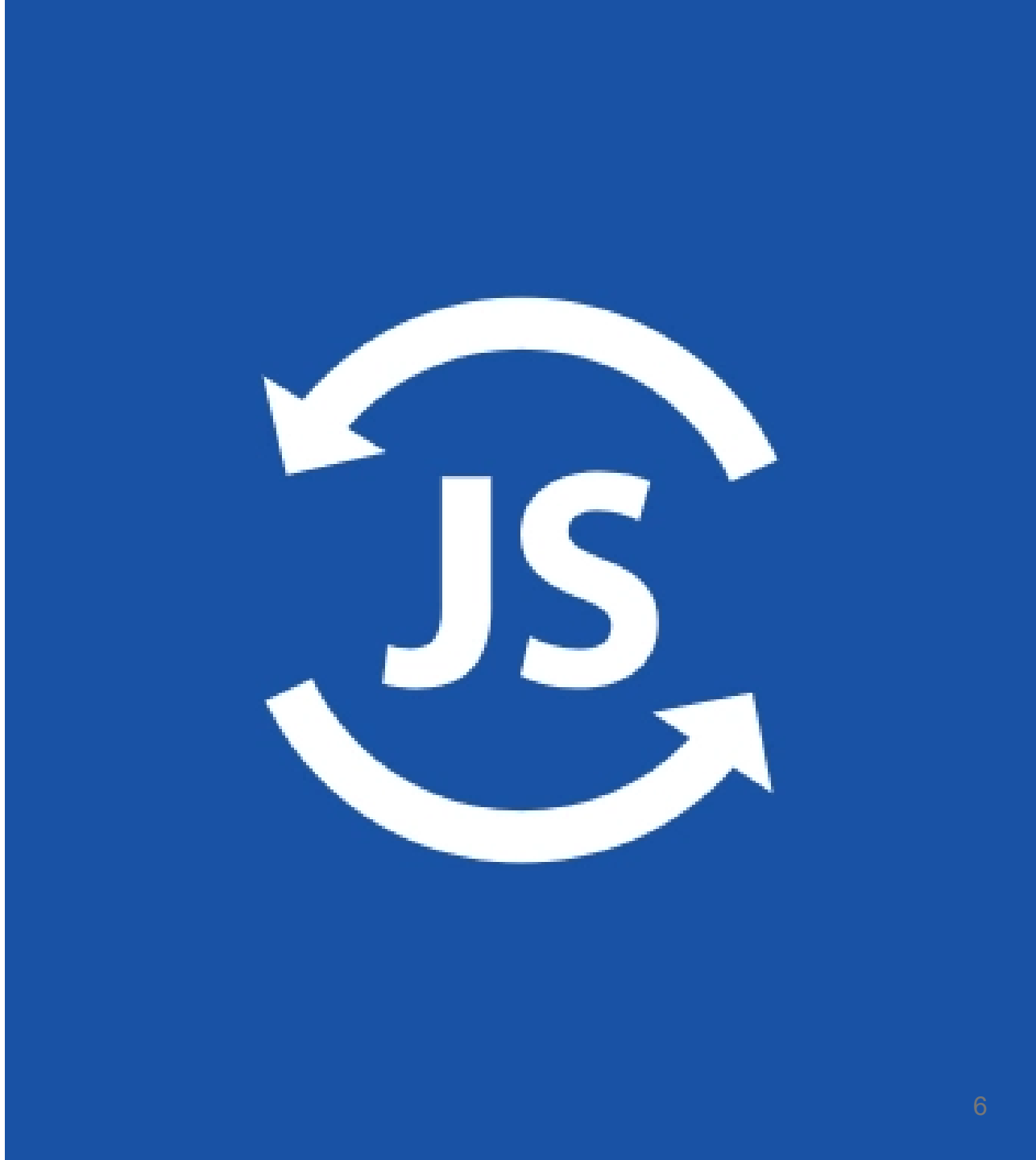
Ressources

MDN docs

47N article, difference between arrow function and normal function

Medium article, cheatsheet arrow function

Asynchronous code



Commençons par le commencement.

Un langage de programmation est une langue qui permet de converser avec la machine. Néanmoins la machine finira toujours par parler en binaire et l'humain en anglais (l'anglais ici pour faciliter la démonstration). Il a donc fallu trouver un juste milieu => tous les langages de prog.

Parmi ces langages il y a JavaScript. Comme écrit dans son nom, JS est un langage de script et il doit donc être interprété. Cet interpréteur (engine in english) s'appelle ChromeV8. Cet interpréteur est incorporé à l'interne de tous les browser modernes. C'est pour cela que JS est un de rares langages de programmation a être compris nativement par les browser.

Caractéristiques de JS

Il faut savoir que le code JS normal est dit « synchrone », car il exécute ses directives (= ce qu'on lui demande de faire) les unes après les autres.

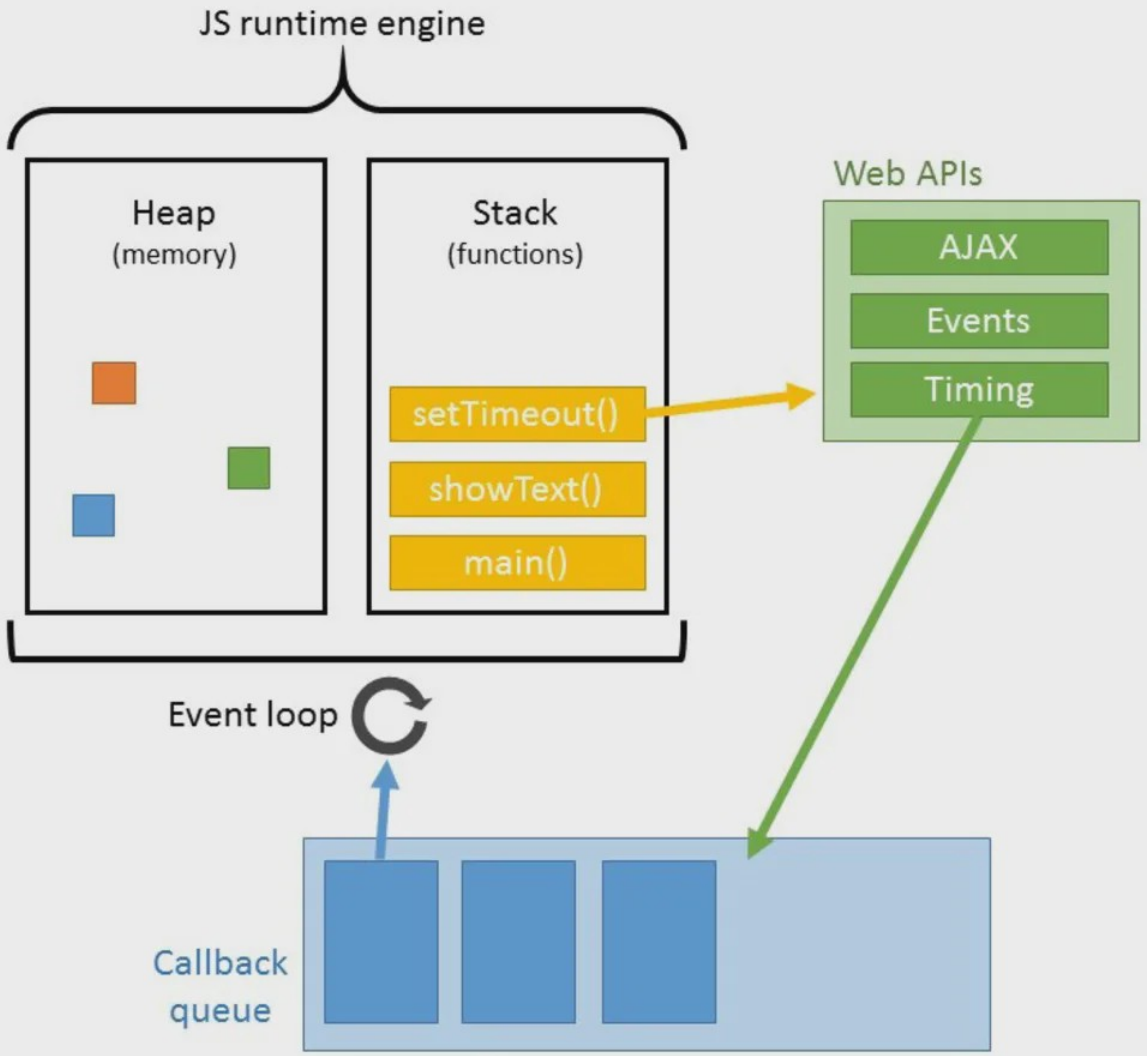
```
console.log("Hello World");
console.log(2+3);

//output
Hello world
5
```

Jusque là : normal.

Sauf que JavaScript est bien plus puissant que ça et cela grâce à son code asynchrone et à sa boucle d'événement (event loop).

Event loop et code asynchrone

À l'interne du runtime engine (ChromeV8) il y a les fonctions natives à ChromeV8 (exemple console.log). Cependant ChromeV8 va pouvoir se connecter à ce qu'on appelle des Web APIs qui elles seront asynchrones. Pour que cela marche, ChromeV8 va envoyer la fonction ne faisant pas partie du cœur natif de l'engine (exemple setTimeout()) à la Web API dédiée. Quand la Web API aura finie de calculer la fonction elle va l'envoyer à la callback queue. Là rentre en jeu l'event loop car son « seul » rôle sera d'envoyer les éléments de la queue à la stack.



Les promesses

Les promesses sont la suite logique au code asynchrone, car elles seront **toujours asynchrones**. Elles vont comme leur nom l'indique vous faire une promesse de réponse. Vous recevrez une de ces trois réponses c'est certain.

- Pending, la promesse est en attente
- Resolved, la promesse est bien résolue
- Rejected, la promesse n'a pas été acceptée

La compréhension des promesses est surtout très utile, car elles seront utilisées lors d'appels asynchrones et lors de l'utilisation de divers Web API pour construire notre PWA.

async / await

Les mots clefs async et await sont un moyen d'orthographier notre code quand une utilise des promesses (et donc du code asynchrone et donc l'event loop, tout est lié 😄)

```
//async est mis au début de ma fonction pour avertir que celle-ci est asynchrone
const fetchGET = async (url) => {
  const myinit = { method: "GET" };

  try {
    //le try and catch est ici une façon de récuérer l'erreur si les promesses ont une erreur
    //je peux ensuite enchainer les await pour appeler différentes promesses
    const response = await fetch(url, myinit); //ici une prommesse utilisant la WEB API fetch
    const responseJSObject = await response.json(); //ici la même chose mais avec la WEBP API json
    return responseJSObject; //je retourne ma réponse "si tout c'est bien passé"
  } catch (err) {
    //je rattrape l'erreur seuleument si il y en a une
    return err; //je retourne cette réponse à la place de ma réponse "si tout c'est bien passé"
  }
};
```

Ressources

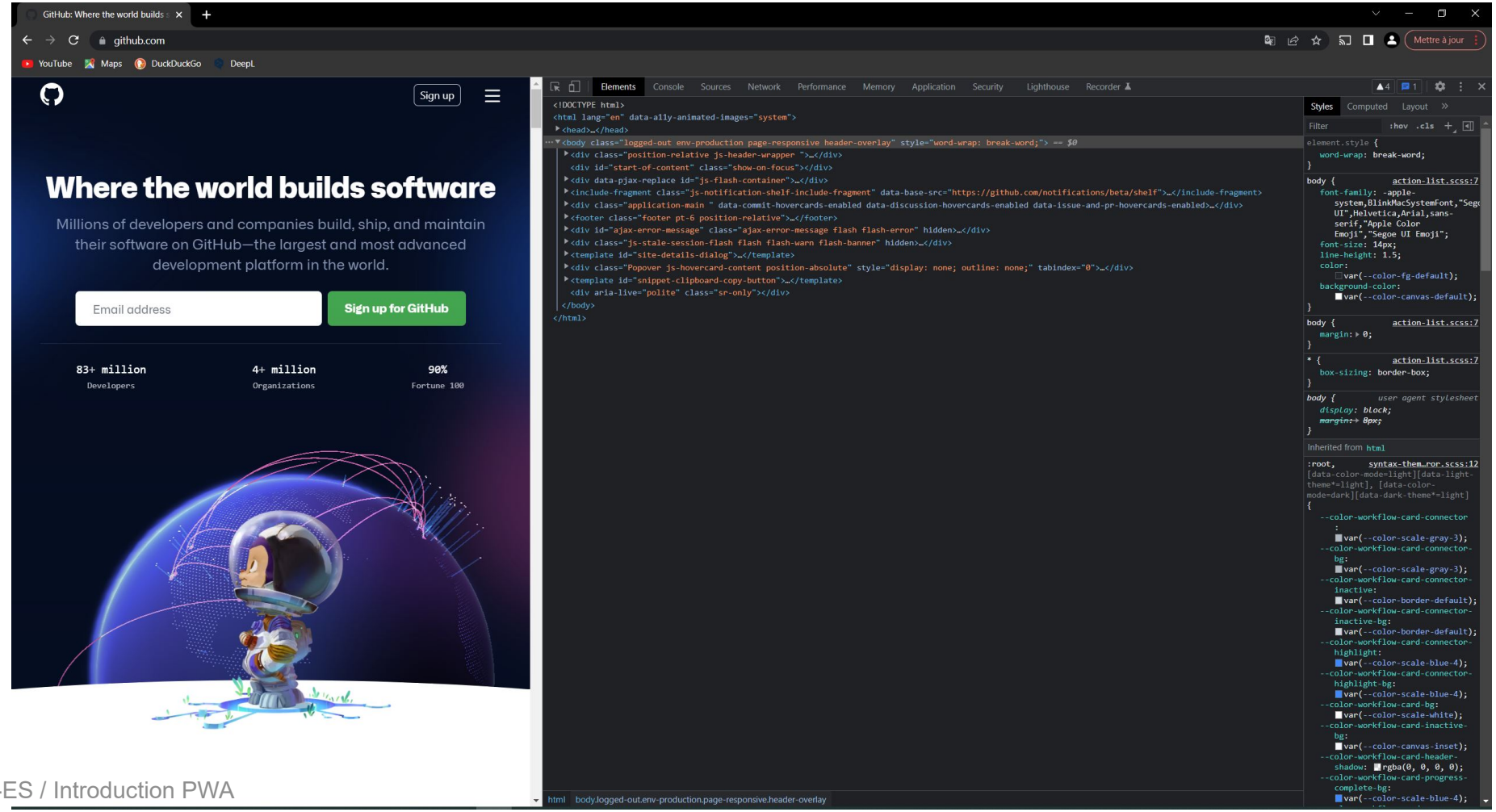
Complément au cours : asynchronicité de JS + promesses + asnyc/await

Boucle d’événement, que diable est-ce au fait ? Vidéo YouTube

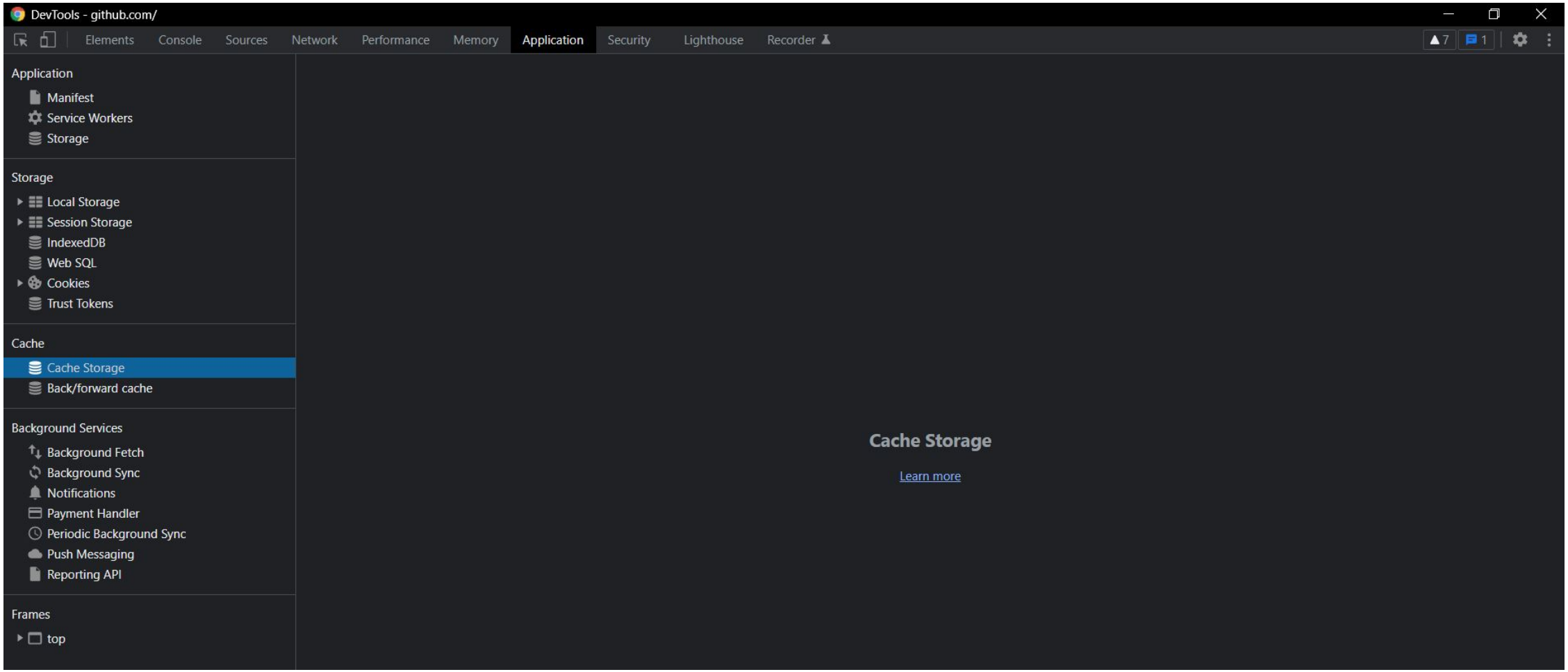
Event loop, MDN

Outils de débogage

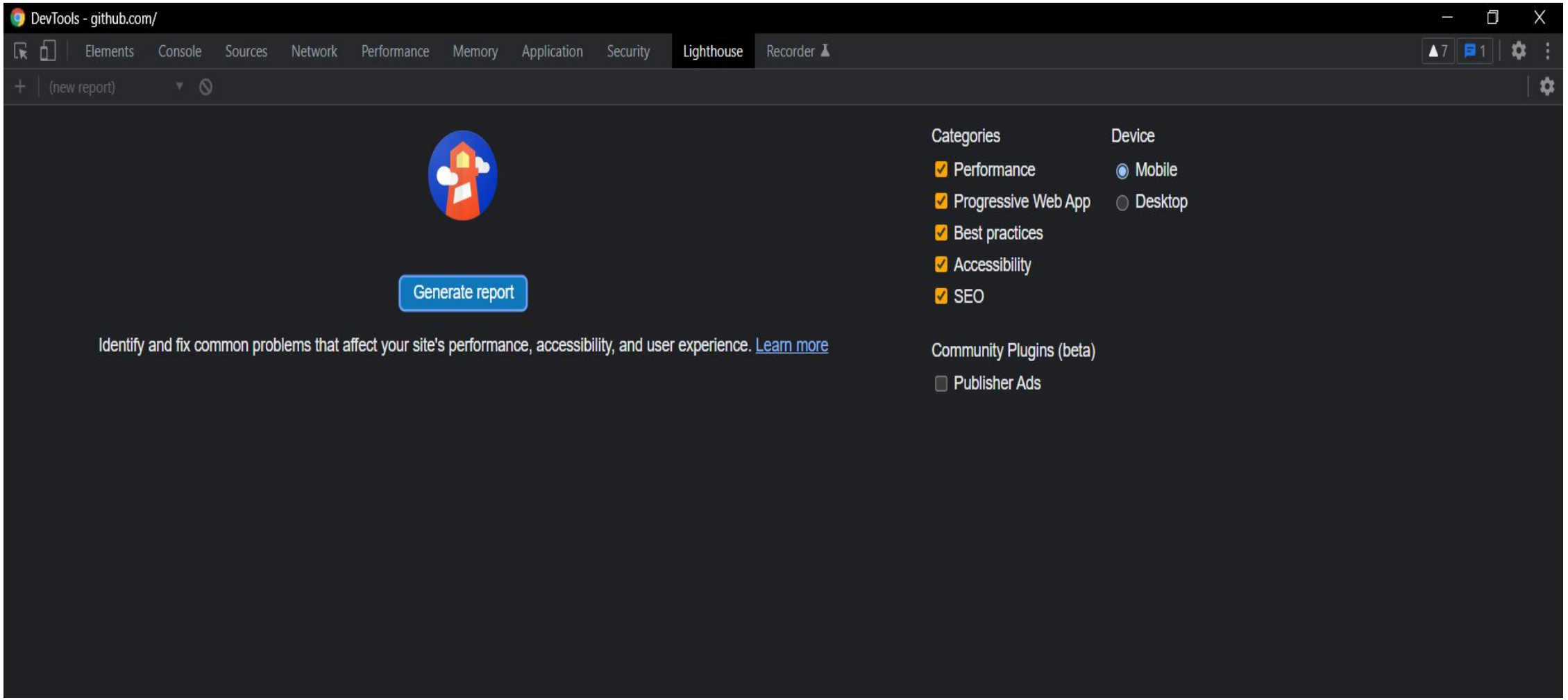
Les outils de débogage sont tous présents dans la section inspection de votre browser (ici Chrome pour la démonstration).
Pour la trouver => clic droit => inspecter.



La section « application » nous permettra d'utiliser la base de données côté client, le cache et le local storage (des noms qui vous seront expliqués plus tard)



La section « Lighthouse » (seulement disponible sur Chrome) nous permettra de faire des audits de note PWA pour savoir si elle est cotée selon les normes en vigueurs.



Ressources

Chrome outil de développement