Découvrir  fetch !

L'API fetch fournit une interface Javascript pour accéder et manipuler les demandes et les réponses sur un réseau.

Elle est fournie par la méthode globale window.fetch() du navigateur.

🪛Mise en place

Créer un fichier index.html

index.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3.
4. <head>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.   <title>Document</title>
```

```
</head>

<body>
  <div class="container"></div>
  <script type="module" src="app.mjs"></script>
</body>

</html>
```

Créez un module  quizz.mjs

 quizz.mjs

```
let quizz;

(async () => {
  const url = "https://opentdb.com/api.php?amount=100";
  const response = await fetch(url);
  quizz = await response.json();
})();

export { quizz };
```

Une autre écriture (sans l'exécution immédiate de la fonction anonyme) serait

 quizz.mjs

```
let quizz;

const run = async () => {
```

```
    const url = "https://opentdb.com/api.php?amount=100";
    const response = await fetch(url);
    quizz = await response.json();
};

run();

export { quizz };
```

Nous allons faire appel au module quizz.mjs au travers du fichier app.mjs.

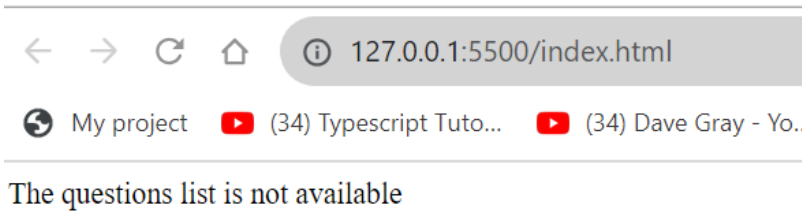
Créez un fichier app.mjs

app.mjs

```
import { quizz } from "./quizz.mjs";

function render(quizz) {
  if (!quizz) {
    throw "The questions list is not available";
  }

  const list = quizz.results
    .map((question) => {
      return `<li
    data-answer="${question.correct_answer}">${question.question} </li>`;
    })
    .join("");

```

```
14.  return `<ol>${list}</ol>`;
15.}
16.
17.const container = document.querySelector(".container");
18.try {
19.  container.innerHTML = render(quizz);
20.} catch (e) {
21.  container.innerHTML = e;
22.}
```

Lancez live-server  index.html



💡 Analyse :

Il semble que le module principal n'attend pas la réponse du fetch !

Correctif !

Pour résoudre ce problème, vous pouvez exporter **une promesse** à partir du module quizz.mjs et attendre que l'appel à l'API soit terminé avant d'utiliser son résultat.

Modifiez le module  quizz.mjs

quizz.js

```
let quizz;

export default (async () => {
  const url = "https://opentdb.com/api.php?amount=100";
  const response = await fetch(url);
  quizz = await response.json();
})();

export { quizz };
```

Modifiez le module app.mjs.

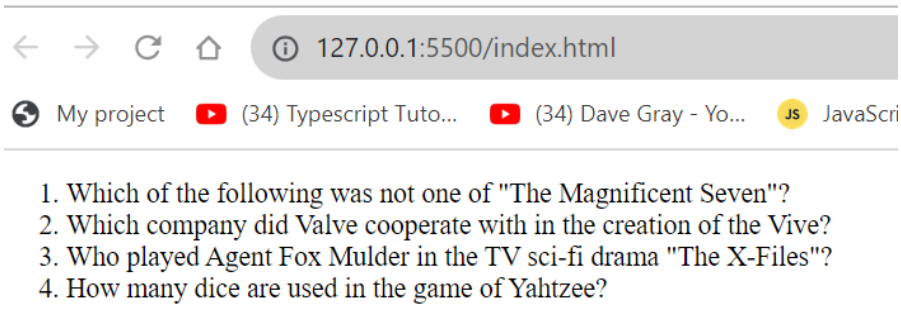
app.js

```
import promise, { quizz } from "./quizz.mjs";

function render(quizz) {
  if (!quizz) {
    throw "The questions list is not available";
  }

  const list = quizz.results
    .map((question) => {
      return `<li
data-answer="${question.correct_answer}">${question.question} </li>`;
    })
    .join("");

  return `<ol>${list}</ol>`;
```

```
15.}
16.
17.promise.then(() => {
18.  let container = document.querySelector(".container");
19.  try {
20.    container.innerHTML = render(quizz);
21.  } catch (error) {
22.    container.innerHTML = error;
23.  }
24.});
```

Lig. 1 : Le module par défaut peut être nommé comme l'on veut. On fait apparaître ici clairement une promesse.

🥷Lancez live-server sur le fichier  index.html. Vous devriez obtenir le quiz suivant.



🆒 Améliorations pour le fun

Pour voir les réponses, nous ajoutons un fichier de style  style.css

 style.css

```
1.  li:hover::after {
```

```
    content: " 🖉 " attr(data-answer)" ";
}

li:hover {
    background: black;
    color: white;
}
```

Lig.2 : On recherche l'attribut data-answer pour l'afficher au passage de la souris sur la question !

Pensez à ajouter le lien sur le fichier  style.css dans le fichier  index.html

 index.html

```
...
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style.css">
</head>
...d
```


LIGHTYEAR

Et oui,  ES2022 est là pour nous simplifier la tâche !

En particulier, nous allons utiliser le **" top-level await".**

🆘Dans un module, vous pouvez utiliser le mot-clé await sans placer une déclaration à l'intérieur d'une fonction asynchrone.

Modifiez le module quizz.mjs.

quizz.mjs

```
1. const url = "https://opentdb.com/api.php?amount=100";
2. const response = await fetch(url);
3. let quizz = await response.json();
4.
5. export { quizz };
```

Modifiez le module app.mjs

app.js

```
1. import { quizz } from "./quizz.mjs";
2.
3. function render(quizz) {
4.   if (!quizz) {
5.     throw "The questions list is not available";
6.   }
7.
```

```
8.    const list = quizz.results
9.      .map((question) => {
10.      return `<li
    data-answer="${question.correct_answer}">${question.question} </li>`;
11.    })
12.    .join("");
13.
14.  return `<ol>${list}</ol>`;
15.}
16.
17.let container = document.querySelector(".container");
18.try {
19.  container.innerHTML = render(quizz);
20.} catch (error) {
21.  container.innerHTML = error;
22.}
```

Le module app.mjs attendra que le fetch dans le module quizz.mjs soit terminé avant de s'exécuter.

Un module await de premier niveau agit comme une fonction asynchrone.

Voici le code asynchrone en action !

🌎https://dupontdenis.github.io/fetch-es20/

Annexe.

Vous pouvez si vous n'aimez pas les fonctions IIFE (Immediately Invoked Function Expression) écrire les modules différemment. Voici une nouvelle écriture de nos modules.

quizz.mjs

```
let quizz;

export default async () => {
  const url =
"https://opentdb.com/api.php?amount=100";
  const response = await fetch(url);
  quizz = await response.json();
};

export { quizz };
```

app.mjs

```
import promise, { quizz } from "./quizz.mjs";

function render(quizz) {
  if (!quizz) {
    throw "The questions list is not available";
  }

  const list = quizz.results
    .map((question) => {
      return `<li
data-answer="${question.correct_answer}">${que
stion.question} </li>`;
    })
    .join("");

  return `<ol>${list}</ol>`;
}

promise().then(() => {
  let container =
document.querySelector(".container");
  try {
    container.innerHTML = render(quizz);
  } catch (error) {
    container.innerHTML = error;
  }
});
```

Notez que vous écrirez dans app.mjs promise().then()