# Apprendre la syntaxe JavaScript : Requêtes

## Corps de réponse au format JSON

La `.json()` méthode résoudra une promesse retournée en un objet JSON, en analysant le corps du texte en tant que JSON.
Dans l'exemple de code, la `.json()` méthode est utilisée sur l' `response` objet qui renvoie une promesse à un corps de réponse au format JSON en tant que `jsonResponse` .

```
récupérer ( 'url' )
. alors (
  réponse => réponse . json ( )
) . then ( jsonResponse => {
  consoler . log ( jsonResponse ) ;
} ) ;
```

## Requête HTTP GET

Les requêtes HTTP `GET` sont faites dans le but de récupérer des informations ou des données à partir d'une source (serveur) sur le Web.
`GET` Les demandes n'ont pas de *corps* , de sorte que les informations requises par la source, afin de renvoyer la réponse appropriée, doivent être incluses dans le chemin de l'URL de la demande ou la chaîne de requête.

## La `fetch()`fonction

L'API JavaScript Fetch est utilisée pour écrire des requêtes HTTP à l'aide de Promises. La fonction main `fetch()` accepte un paramètre d'URL et renvoie une promesse qui se résout en un objet de réponse ou la rejette avec un message d'erreur si une erreur réseau se produit.
L'exemple de code commence par appeler la `fetch()` fonction. Ensuite, une `then()` méthode est chaînée à la fin du `fetch()` . Il se termine par le rappel de réponse pour gérer le succès et le rappel de rejet pour gérer l'échec.

```
récupérer ( 'url' )
. alors (
  réponse => {
    consoler . log ( réponse ) ;
  } ,
 rejet => {
    consoler . erreur ( rejet . message )
;
) ;
```

## Personnalisation des requêtes de récupération

La `fetch()` fonction accepte un deuxième argument facultatif, un objet options, utilisé pour personnaliser la requête. Cela peut être utilisé pour modifier le type de demande, les en-têtes, spécifier un corps de demande, et bien plus encore.

```
fetch('https://api-to-call.com/endpoint',
{
  method: 'POST',
  body: JSON.stringify({id: "200"})
```

Dans l'exemple de code ci-dessous, la `fetch()` fonction comme second argument, un objet contenant des options pour la requête d'extraction en spécifiant le `method` et le `body`.

```
}).then(response => {
  if(response.ok){
          return response.json();
  }
        throw new Error('Request
failed!');
}, networkError => {
  console.log(networkError.message);
}).then(jsonResponse => {
  console.log(jsonResponse);
})
```

## HTTP POST Request

HTTP `POST` requests are made with the intention of sending new information to the source (server) that will receive it.
For a `POST` request, the new information is stored in the *body* of the request.

## Using `async...await` with Fetch

The `async ... await` syntax is used with the Fetch API to handle promises.
In the example code, the `async` keyword is used to make the `getSuggestions()` function an async function. This means that the function will return a promise. The `await` keyword used before the `fetch()` call makes the code wait until the promise is resolved.

```
const getSuggestions = async () => {
  const wordQuery = inputField.value;
  const endpoint =
`${url}${queryParams}${wordQuery}`;
  try{
const response = await fetch(endpoint,
{cache: 'no-cache'});
    if(response.ok){
      const jsonResponse = await
response.json()
    }
  }
  catch(error){
    console.log(error)
  }
}
```