

# **La programmation AJAX**

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. XMLHttpRequest</b>	<b>3</b>
<b>III. Exercice : Appliquez la notion</b>	<b>6</b>
<b>IV. Créer des requêtes AJAX</b>	<b>7</b>
<b>V. Exercice : Appliquez la notion</b>	<b>10</b>
<b>VI. Les headers HTTP</b>	<b>12</b>
<b>VII. Exercice : Appliquez la notion</b>	<b>14</b>
<b>VIII. Auto-évaluation</b>	<b>16</b>
A. Exercice final.....	16
B. Exercice : Défi.....	18
<b>Solutions des exercices</b>	<b>19</b>

## I. Contexte

**Durée :** 2 h

**Environnement de travail :** Visual Studio Code

**Pré-requis :** Bases de JavaScript, connaissance de JSON et savoir interagir avec une API

### Contexte

Dans une application web dynamique et moderne, il est essentiel de pouvoir modifier ou mettre à jour les données affichées, sans pour autant recharger toute la page.

AJAX (ou *Asynchronous JavaScript And XML*) est un ensemble de technologies qui vont permettre des échanges rapides de données entre client et serveur, sans rechargement de la page HTML.

Pour le web, le cas d'utilisation le plus simple est que le client (le navigateur) envoie une requête http au serveur. Lorsque le serveur répond, le client affiche la réponse (souvent au format HTML) à l'utilisateur.

Utiliser AJAX permet de demander au navigateur d'envoyer une requête http depuis javascript.

Contrairement au comportement "classique" du navigateur, la réponse à cette requête doit être traitée dans le javascript. Ce qui permet par exemple, de ne mettre à jour qu'une partie du DOM avec des données qui viennent d'être reçues en réponse à une requête, ou d'envoyer des données (d'un formulaire ou autre) sans pour autant recharger tout le contenu de la page.

Comme il est impossible de savoir "si" et "quand" un serveur va répondre à une requête on utilise dans la plupart des cas AJAX de manière asynchrone. Une fois la requête envoyée, le code javascript continue de s'exécuter.

Pour le traitement de la réponse, il va falloir créer une fonction de traitement qui sera exécutée lorsque la réponse arrivera.

Il faut donc ajouter un listener à notre requête pour qu'il appelle notre fonction de traitement au bon moment.

## II. XMLHttpRequest

### Objectifs

- Comprendre le fonctionnement de l'objet XMLHttpRequest
- Connaître les fonctionnalités principales de l'objet XMLHttpRequest

### Mise en situation

Nous allons présenter ici les principales fonctions de l'objet XMLHttpRequest.

### Syntaxe

#### Initialisation de la requête

Pour initialiser une requête, il faut d'abord créer une instance de l'objet XMLHttpRequest.

La méthode `.open()` permet l'initialisation de la requête.

Elle respecte la méthode HTTP c'est-à-dire qu'elle permet d'envoyer des requêtes GET ou POST (ou HEAD) classique. Avec `open()` on peut aussi paramétrer une URL, passer des données venant d'une base de données comme par exemple un login, un mot de passe, etc ... tout ça de manière asynchrone.

La requête est envoyée avec la méthode `.send()`.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://reqres.in/api/users?page=1');
3 xhr.send();
```

### Attention

Par défaut, la requête est asynchrone, c'est-à-dire que le script continue à être exécuté, sans attendre la réponse qui sera retournée via un `callback`.

C'est un comportement souhaitable dans l'immense majorité des cas. Cependant, il est possible de forcer un comportement synchrone en passant la valeur `false` au paramètre asynchrone dans l'appel de la méthode.

```
1 xhr.open('get', 'https://reqres.in/api/users?page=1', false);
```

### Les différents états

La requête envoyée étant asynchrone, cela signifie qu'elle est exécutée en parallèle avec d'autres fonctions. Il est important d'avoir une fonction de `callback` déclenchée à la fin du traitement pour récupérer le résultat de la requête.

En JavaScript, un `callback` permet à une fonction d'appeler une autre fonction. Il s'agit plus précisément d'une fonction de rappel qui peut être exécutée après la fin d'une autre fonction.

Pour cela, XHR possède un événement `readystatechange` qui est déclenché à chaque changement d'état de l'instance.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://reqres.in/api/users');
3
4 // readystatechange sera appelée chaque fois que le readyState changera. Ce qui permettra de
  // générer un callback.
5 xhr.addEventListener('readystatechange', function() {
6   // readyState indiquera l'état de la requête de l'objet XMLHttpRequest.
7   console.log("Current XHR state = " + xhr.readyState);
8 });
9
10 xhr.send();
```

Valeur	Etat	Description
0	UNSENT	L'objet XHR est créé, mais pas encore initialisé via la méthode <code>.open()</code>
1	OPENED	L'objet XHR est initialisé, mais la requête n'est pas envoyée via la méthode <code>.send()</code>
2	HEADERS_RECEIVED	La requête est envoyée au serveur et le <code>HEADER</code> de la réponse a été reçu
3	LOADING	Le transfert de données est en cours
4	DONE	Le transfert de données est terminé, l'opération est complète

Il suffit donc de tester l'état et de récupérer la réponse quand la requête est terminée.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', "https://reqres.in/api/users");
3
4 xhr.addEventListener('readystatechange', function() {
5     if(xhr.readyState === 4) {
6         console.log("Response = " + xhr.response);
7     };
8 });
9
10 xhr.send();
```

## La réponse

Si la requête est complète et réussie, le retour du serveur (*response*) sera accessible via la propriété `.response` de l'objet XHR.

La propriété `.status` de XHR nous renseigne sur le code HTTP de cette réponse.

Voici les plus courants :

- 200 : Succès de la requête
- 301 : Redirection
- 403 : Accès refusé
- 404 : Adresse non trouvée
- 500 : Erreur serveur

On pourra donc tester l'état de la requête avant d'essayer de lire sa réponse.

### Exemple Requête vers une API retournant une liste de personnes

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://reqres.in/api/users?page=2');
3
4 xhr.addEventListener('readystatechange', function() {
5     if(xhr.readyState === 4 && xhr.status === 200) {
6         console.log("Response = " + xhr.response);
7     };
8 });
9
10 xhr.send();
```

Résultat en console :

```
1 Response = {"page":2,"per_page":6,"total":12,"total_pages":2,"data":
[{"id":7,"email":"michael.lawson@reqres.in","first_name":"Michael","last_name":"Lawson","avatar":"https://
{"id":8,"email":"lindsay.ferguson@reqres.in","first_name":"Lindsay","last_name":"Ferguson","avatar":"https
{"id":9,"email":"tobias.funke@reqres.in","first_name":"Tobias","last_name":"Funke","avatar":"https://s3.am
{"id":10,"email":"byron.fields@reqres.in","first_name":"Byron","last_name":"Fields","avatar":"https://s3.a
{"id":11,"email":"george.edwards@reqres.in","first_name":"George","last_name":"Edwards","avatar":"https://
{"id":12,"email":"rachel.howell@reqres.in","first_name":"Rachel","last_name":"Howell","avatar":"https://s3
{"company":"StatusCode Weekly","url":"http://statuscode.org/","text":"A weekly newsletter
focusing on software development, infrastructure, the server, performance, and the stack end
of things."}]}
```

### Remarque Format JSON

Le format de chaîne de caractères récupérée dans l'exemple précédent est au format JSON, pour JavaScript Object Notation. Il s'agit d'une notation permettant de représenter des informations structurées, ressemblant beaucoup à ce qu'on peut écrire en JavaScript. Nous n'allons pas nous attarder ici en détail sur ce format, mais sachez qu'il est possible d'analyser cette chaîne de caractère et de la transformer en la valeur JavaScript décrite par celle-ci grâce à l'objet `JSON` et sa méthode `parse()`. Un cours est dédié à ce format.

```
1 const json = '{"page": 2, "per_page": 6}'; // C'est la représentation d'un objet JavaScript en
  JSON
2 const object = JSON.parse(json);
3
4 console.log(object); // Affiche {page: 2, per_page: 6}
5 console.log(object.page); // Affiche 2
```

### Exemple

Pour en revenir à notre exemple, voici comment convertir la réponse reçue en JSON en un objet JavaScript :

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://reqres.in/api/users?page=2');
3
4 xhr.addEventListener('readystatechange', function() {
5     if(xhr.readyState === 4 && xhr.status === 200) {
6         const data = JSON.parse(xhr.response);
7     };
8 });
9
10 xhr.send();
```

### Syntaxe À retenir

1. `XMLHttpRequest` est l'objet qui nous permet de créer, envoyer et obtenir le résultat d'une requête HTTP.
2. XHR est initialisé avec la méthode `.open()` et la requête est lancée avec la méthode `.send()`.
3. Les propriétés `.readyState` et `.status` nous renseignent respectivement sur l'état d'avancement et sur l'état de réussite de la requête.
4. Une fois complète, le corps de la réponse est accessible dans la propriété `.response`.

### Complément

MDN : `XMLHttpRequest`<sup>1</sup>

## III. Exercice : Appliquez la notion

### Question

[solution n°1 p.21]

La page de votre site affiche une liste d'utilisateurs dans un tableau. On obtient cette liste via une requête avec la méthode GET : `'https://reqres.in/api/users?page=1'`.

Initialisez l'objet `XmlHttpRequest` pour récupérer la liste des utilisateurs.

La propriété `data` de l'objet retourné stocke la liste des utilisateurs.

Vous devrez ajouter la propriété `name` à chaque item de la liste, en concaténant les propriétés `first_name` et `last_name`.

<sup>1</sup> <https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>

Retournez en console la liste des utilisateurs en ayant ajouté au préalable la propriété `name` à chaque utilisateur. Vous utiliserez un environnement de travail `repl.it`<sup>1</sup>.

### Indice :

N'oubliez pas d'utiliser la méthode `JSON.parse()` pour transformer l'objet JSON en objet JavaScript.

## IV. Créer des requêtes AJAX

### Objectifs

- Comprendre l'utilité d'une requête AJAX
- Savoir récupérer et envoyer des données avec `XMLHttpRequest`
- Savoir quel verbe HTTP utiliser

### Mise en situation

Pour dynamiser son site web, il est fréquent de vouloir échanger des données avec un serveur et que cela soit transparent pour l'utilisateur, c'est-à-dire sans recharger la page entièrement.

**AJAX** (*Asynchronous JavaScript And XML*) va nous permettre d'envoyer des requêtes HTTP en arrière-plan, au travers de l'objet `XMLHttpRequest`.

#### Rappel XMLHttpRequest

`XMLHttpRequest` (XHR) est un objet du navigateur permettant l'envoi de requêtes HTTP en asynchrone. Il est aujourd'hui implémenté par tous les navigateurs web modernes.

Pour l'utiliser, nous devons créer une instance de l'objet `XMLHttpRequest`, l'initialiser avec la méthode `.open()` et envoyer la requête avec la méthode `.send()`.

Étant asynchrone, il est nécessaire d'ajouter une fonction de `callback` à l'événement `readystatechange` de l'instance XHR, pour traiter le retour de la requête.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'http://mon-serveur/ma-ressource');
3
4 xhr.addEventListener('readystatechange', function() {
5     if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
6         console.log("Response = " + xhr.response);
7     };
8 });
9
10 xhr.send();
```

### Méthodes de requêtes HTTP

Le protocole HTTP définit un ensemble de méthodes pour qualifier l'action souhaitée sur la ressource.

Ces méthodes, appelées aussi **Verbe HTTP**, sont normées.

Voici les principales :

- `GET` : afin de récupérer des données.
- `HEAD` : réponse identique à `GET`, sauf que seul l'en-tête de réponse sera renvoyé (le corps de la réponse sera omis). Utile pour vérifier la présence d'une ressource.

---

<sup>1</sup> <https://repl.it/repls/FewNotableCookie>

- **POST** : pour envoyer des données au serveur (création d'une entité).
- **PUT** : pour envoyer des données au serveur (modification d'une entité).
- **DELETE** : pour supprimer une entité.

## La méthode GET

Pour récupérer une ressource, nous utiliserons la méthode **GET**.

Si l'on veut récupérer la liste de tous les utilisateurs, la méthode **GET** est idéale.

```
1 const xhr = new XMLHttpRequest();
2 const verb = 'GET';
3 const route = 'http://mon-serveur/utilisateurs';
4 let result;
5
6 xhr.open(verb, route);
7
8 xhr.addEventListener('readystatechange', function() {
9     if(xhr.readyState === 4 && xhr.status === 200) {
10         result = xhr.response;
11     };
12 });
13
14 xhr.send();
```

Si, maintenant, nous voulons récupérer uniquement les utilisateurs dont le prénom est **François**, la méthode **GET** est toujours indiquée. Mais il faudra ajouter des paramètres à notre requête.

Pour éviter les erreurs dues aux caractères spéciaux, il est recommandé d'utiliser la fonction JavaScript `encodeURIComponent()`.

```
1 const route = 'http://mon-serveur/utilisateurs?firstname=' + encodeURIComponent('François');
```

## Les méthodes POST et PUT

Pour créer et modifier une entité, les méthodes **POST** et **PUT** sont indiquées.

Leurs fonctionnements sont identiques, si ce n'est que, pour la méthode **PUT**, un paramètre est nécessaire pour identifier la ressource à modifier au niveau du serveur (comme un ID unique, par exemple).

Pour envoyer les paramètres, nous n'utiliserons pas l'URL comme avec la méthode **GET**, mais nous ajouterons ces paramètres à la méthode `.send()`.

Pour spécifier au serveur que des paramètres vont être envoyés par la méthode `.send()`, il faut ajouter une information dans le `header` de la requête HTTP.

```
1 const xhr = new XMLHttpRequest();
2 const verb = 'POST';
3 const route = 'http://mon-serveur/utilisateurs';
4
5 xhr.open(verb, route);
6
7 xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
8
9 xhr.send('name=Marty&firstname=David&city=Perpignan');
```

Une autre méthode, plus simple et plus évoluée, consiste à utiliser un objet `FormData`.



```

1 const xhr = new XMLHttpRequest();
2 const verb = 'POST';
3 const route = 'http://mon-serveur/utilisateurs';
4
5 xhr.open(verb, route);
6
7 var form = new FormData();
8 form.append('name', 'Marty');
9 form.append('firstname', 'David');
10 form.append('city', 'Perpignan');
11
12 xhr.send(form);

```

Cet objet `FormData` pourra même servir pour l'upload de fichiers.

### Les erreurs

Il est important de gérer les erreurs possibles lors de l'envoi ou de la réception de requêtes HTTP lorsque l'on fait de l'AJAX.

```

1 var xhr = new XMLHttpRequest();
2 var verb = 'GET';
3 var route = 'http://mon-serveur/utilisateurs';
4 var result;
5
6 xhr.open(verb, route);
7
8 xhr.addEventListener('readystatechange', function() {
9     if(xhr.readyState === 4) {
10         //Si le status n'est pas 200 (HTTP.OK), on alerte l'utilisateur.
11         if(xhr.status !== 200) {
12             alert('An error occurred. Code: ' + xhr.status + ', Message : ' + xhr.statusText);
13         } else {
14             result = xhr.response;
15         }
16     }
17 });
18
19 xhr.send();

```

Une erreur ne sera pas forcément retournée par un code HTTP : `XmlHttpRequest` peut lui-même lever certaines erreurs.

Pour les traiter, il faudra utiliser un callback sur l'événement `'error'` de l'objet `XmlHttpRequest`.

```

1 var xhr = new XMLHttpRequest();
2 var verb = 'GET';
3 var route = 'http://mon-serveur/utilisateurs';
4 var result;
5
6 xhr.open(verb, route);
7
8 xhr.addEventListener('error', function() {
9     alert('An error occurred.');
```

### Conseil L'API REST

Rest est un ensemble de recommandations auxquelles un développeur doit adhérer afin de pouvoir considérer une API comme étant « RESTful ». Les API REST tirent parti des méthodes HTTP vues plus haut dans ce cours. Les applications côté client de l'API utilisent des appels HTTPS pour :

- Appeler une ressource avec une méthode GET,
- Transmettre des données au serveur avec une méthode POST,
- DELETE pour la supprimer.

Ces trois méthodes HTTPS sont les plus utilisées par les API RESTful mais il est possible de recourir à d'autres méthodes comme HEAD, PUT, PATCH, CONNECT, OPTION ainsi que TRACE.

De nombreux sites font appel à des API RESTful, comme Google, Amazon et Twitter.

### Syntaxe À retenir

- AJAX est un ensemble de technologies permettant de lancer des requêtes HTTP en arrière-plan, entre autres à l'aide de l'objet XMLHttpRequest (XHR).
- Nous pourrions ainsi créer, lire, mettre à jour, et supprimer des données. En anglais : *Create, Read, Update, Delete (C.R.U.D.)*.
- XHR nous donne accès à un éventail de méthodes et de propriétés qui permettront de gérer finement les données et les erreurs.

### Complément

MDN : Méthodes de requête HTTP<sup>1</sup>

MDN : XMLHttpRequest<sup>2</sup>

MDN : FormData<sup>3</sup>

## V. Exercice : Appliquez la notion

### Question

[solution n°2 p.21]

L'administrateur d'un site scolaire doit ajouter les nouveaux élèves de l'établissement au logiciel de gestion. Pour cela, il utilise une interface qui lui permet de rentrer le nom de l'élève, son prénom, et sa classe.

Informaticquement, il envoie au serveur un objet possédant les propriétés `firstName`, `lastName`, `grade` et `numberClass`.

La propriété `firstName` correspond au prénom de l'élève, `lastName` au nom de famille, `grade` au niveau et `numberClass` au numéro de la classe.

Exemple : pour Marie Dupond, en 6<sup>ème</sup> 5 :

```
1 {
2   id: '',
3   firstName: 'Marie',
4   lastName: 'Dupond',
5   grade: 6,
6   numberClass: 5
7 }
```

1 <https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thodes>

2 <https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>

3 <https://developer.mozilla.org/fr/docs/Web/API/FormData/FormData>

8

Lorsque l'on clique sur le bouton **Enregistrer**, le code appelle la fonction `sendForm()`.

À l'aide du code ci-dessous, créez la fonction `sendForm()` qui contiendra une requête AJAX, dont les caractéristiques sont :

- L'API pour enregistrer l'étudiant est `https://reqres.in/api/users`.
- Comme nous envoyons et créons un nouvel objet, c'est une méthode `POST` qui sera utilisée.
- Le statut d'une réponse valide est 201.
- Si la réponse est valide, alors ajoutez l'étudiant au tableau de la page HTML ci-dessous, ainsi que dans le tableau JavaScript `students`.
- Si la réponse n'est pas valide, retournez une alerte avec un message d'erreur.

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet"
6     href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
7     integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZLL5MYx7Ffc+NcPb1dKGj7Sk"
8     crossorigin="anonymous">
9   <title>hour</title>
10 </head>
11 <body>
12   <div class="container-fluid">
13     <div class="row no-gutters justify-content-between">
14       <div class="col-3">
15         <form>
16           <div class="form-group">
17             <label for="firstName">Nom</label>
18             <input type="text" name="firstName" class="form-control" id="firstName">
19           </div>
20           <div class="form-group">
21             <label for="lastName">Prénom</label>
22             <input type="text" name="lastName" class="form-control" id="lastName">
23           </div>
24           <div class="form-group">
25             <label for="grade">Classe</label>
26             <input type="number" name="grade" class="form-control" id="grade">
27           </div>
28           <div class="form-group">
29             <label for="numberClass">Classe</label>
30             <input type="number" name="numberClass" class="form-control" id="numberClass">
31           </div>
32           <button type="button" class="btn btn-primary" onclick="sendForm(firstName.value,
33             lastName.value, grade.value, numberClass.value)">Enregistrer</button>
34         </form>
35       </div>
36       <div class="col-8">
37         <table class="table table-striped text-center">
38           <thead>
39             <tr>
40               <th>Identifiant</th>
41               <th>Nom</th>
42               <th>Prénom</th>
43               <th>Classe</th>
44               <th>Numero de classe</th>

```

```

42         </tr>
43     </thead>
44     <tbody id="tbody"></tbody>
45 </table>
46 </div>
47 </div>
48 </div>
49 </body>
50
51 <script src="script.js"></script>
52 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
    DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous">
    </script>
53 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
54 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
    integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
    crossorigin="anonymous"></script>
55 </html>

1 const tbody = document.getElementById('tbody'); // élément tbody du tableau
2 const students = []; // liste des étudiants
3
4 // fonction de création de ligne
5 const createLine = (tbody, student) => {
6     const newRow = tbody.insertRow();
7     let counter = 0
8     for (const key in student) {
9         const newCell = newRow.insertCell(counter);
10        // La Méthode createTextNode() permet de créer un noeud de texte
11        const newText = document.createTextNode(student[key]);
12        newCell.appendChild(newText);
13        counter++
14    }
15 }
16
17

```

Vous copierez le code dans un environnement de travail repl.it<sup>1</sup>.

## VI. Les headers HTTP

### Objectifs

- Connaître l'utilité d'un header HTTP
- Savoir utiliser les headers HTTP

### Mise en situation

Avec le protocole HTTP, pour échanger des données entre client et serveur, nous utilisons des requêtes et des réponses. Pour permettre au client comme au serveur de traiter efficacement les données qu'elles transportent, requêtes et réponses fournissent une série d'informations complémentaires contenues dans leurs en-têtes (*header HTTP*).

Un header HTTP est toujours un couple clé/valeur, et est noté <clé>: <valeur>.

<sup>1</sup> <https://repl.it/repls/FewNotableCookie>

Un certain nombre de headers HTTP existent et sont régis par différentes normes. Il est cependant tout à fait possible d'utiliser ses propres headers. Par convention, leurs clés seront préfixées X-. Par exemple : X-API-KEY: AA1452654aa.

### Les headers de requête HTTP

Lors de la préparation d'une requête HTTP, nous allons pouvoir ajouter des informations à celle-ci via son header.

XMLHttpRequest nous fournit la méthode `.setRequestheader()` pour éditer le header de la requête.

`.setRequestheader()` devra être appelé après l'initialisation de la requête via la méthode `.open()` et avant la méthode `.send()`.

`.setRequestheader()` pourra être appelé plusieurs fois, les headers se cumuleront.

```
1 var xhr = new XMLHttpRequest();
2 xhr.open('GET', 'http://mon-serveur/ma-ressource');
3
4 xhr.setRequestHeader('Accept', 'text/html');
5 xhr.setRequestHeader('Accept-Charset', 'utf-8');
6
7 xhr.send();
```

Voici quelques exemples de headers de requête HTTP :

- **Accept** : Donne au serveur une indication sur les types MIME du contenu que le client sera capable de traiter.
- **Accept-Charset** : Donne au serveur une indication sur l'encodage que le client attend pour la réponse.
- **Accept-Language** : Donne au serveur une indication sur les langues acceptées que le client attend pour la réponse (Accept-Language: en-US).
- **Authorization** : Informations pour l'authentification HTTP (Authorization: Bearer mF\_9.BA6f-f.1kLM).
- **Content-Type** : Type MIME du contenu du corps de la requête, utilisé avec les méthodes POST et PUT.

### Les headers de réponse HTTP

Lors de la réception d'une réponse HTTP, nous allons vouloir des renseignements sur les données qu'elle contient dans son corps, afin de définir le meilleur traitement.

- XMLHttpRequest nous fournit plusieurs méthodes afin d'accéder aux headers de la réponse.
- `.getAllResponseheaders()` permet de récupérer l'ensemble des headers de la réponse sous forme de string séparés par des CRLF.
- `.getResponseheader()` permet de récupérer la valeur d'un header en particulier, suivant sa clé.

```
1 var xhr = new XMLHttpRequest();
2 xhr.open('GET', 'http://mon-serveur/ma-ressource');
3
4 xhr.addEventListener('readystatechange', function() {
5     if(xhr.readyState === 4 && xhr.status === 200) {
6         console.log("headers= " + xhr.getAllResponseheaders());
7         //Résultat :
8         //headers= date: Fri, 08 Dec 2017 21:04:30 GMT\r\n
9         //content-encoding: gzip\r\n
10        //connection: keep-alive\r\n
11        //[...]
12
13        console.log("Content encodig value = " + xhr.getResponseheader('content-encoding'));
14        //Résultat :
15        //Content encodig value = gzip
```

```
16     };
17 });
18
19 xhr.send();
```

Voici quelques exemples de headers de réponses HTTP :

- `Content-Encoding` : Type d'encodage de la réponse.
- `Content-Length` : Longueur du corps de la réponse en octets.
- `Content-Disposition` : Donne au client une indication sur où afficher le résultat : dans le navigateur avec la valeur `inline`, ou en déclenchant la boîte de dialogue "Enregistrer sous..." du navigateur avec `attachment` (`Content-Disposition: attachment; filename="monFichier.pdf"`).
- `Content-Type` : Type MIME du contenu du corps de la réponse.

#### Remarque Les headers généraux

Certains headers, comme `Content-Type`, pourront être présents aussi bien dans les requêtes que dans les réponses. Leur utilisation dépendra du contexte dans lequel ils seront utilisés.

#### Syntaxe À retenir

- Les headers HTTP sont présents dans chaque requête et réponse. Ils apportent des informations essentielles sur le contenu et sur le comportement à adopter par le client et le serveur.
- `XMLHttpRequest` fournit plusieurs méthodes permettant de lire et d'écrire le header des requêtes et réponses.

#### Complément

MDN : en-têtes HTTP<sup>1</sup>

## VII. Exercice : Appliquez la notion

### Question

[solution n°3 p.23]

Dans le cadre d'un formulaire de connexion à une application, le serveur retourne un token permettant d'identifier l'utilisateur. Ce token doit être envoyé au serveur à chaque requête et le serveur testera sa validité. Cela permet de s'assurer de l'identité de l'utilisateur.

En règle générale, ce `token` est stocké dans le header avec la clé `Authorization`.

Le code ci-dessous simule le renvoi d'un token. Une fois ce token récupéré, une requête doit être envoyée au serveur pour récupérer une liste d'utilisateurs.

Écrivez la requête qui retournera la liste d'utilisateurs en intégrant le token au header, et retournez la liste des utilisateurs en console :

- Méthode : `GET`.
- URL : `https://reqres.in/api/users?page=1`.
- Statut valide : `200`.

Dans ce code, pour récupérer le token, l'appel à une API est simulé via une méthode `setTimeout()`. Cette méthode indique que le code sera déclenché après que le temps indiqué ait été écoulé. Dans cet exemple, 2 secondes.

<sup>1</sup> <https://developer.mozilla.org/fr/docs/Web/HTTP/Headers>

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZL5MYxPfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
6   <title>Connexion</title>
7   <style>
8     .container-fluid {
9       height: 100vh;
10    }
11  </style>
12 </head>
13
14 <body>
15   <div class="container-fluid">
16     <div class="row no-gutters justify-content-center align-items-center">
17       <div class="col-4">
18         <form>
19           <div class="form-group">
20             <label for="mail">email</label>
21             <input type="text" name="mail" class="form-control" id="mail"
value="eve.holt@reqres.in" disabled>
22           </div>
23           <div class="form-group">
24             <label for="psw">Password</label>
25             <input type="password" name="psw" class="form-control" id="psw"
value="cityslicka" disabled>
26           </div>
27           <button type="button" class="btn btn-primary" onclick="sendForm(mail.value,
psw.value)">Connexion</button>
28         </form>
29       </div>
30     </div>
31   </div>
32 </body>
33
34 <script src="script.js"></script>
35 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfxDz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous">
</script>
36 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
crossorigin="anonymous"></script>
37 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
integrity="sha384-0gVRvuATP1z7JjHLku0U7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
crossorigin="anonymous"></script>
38 </html>

```

```

1 let users = []
2
3 const sendForm = (email, password) => {
4   setTimeout(() => {
5     const token = 'ffghthfzszsz654412dsvcds';
6     // Votre code ici
7   }, 2000)
8 }

```

Vous copierez le code dans un environnement de travail repl.it<sup>1</sup>.

<sup>1</sup> <https://repl.it/repls/FewNotableCookie>

## VIII. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°4 p.24]

Exercice

Quelle méthode HTTP faut-il utiliser pour récupérer une entité existante depuis le serveur ?

- ☐ HEAD
- ☐ GET
- ☐ OBTAIN
- ☐ PUT
- ☐ POST

Exercice

Parmi les éléments ci-dessous, lesquels sont des ordres d'appel valides des méthodes de l'objet XMLHttpRequest ?

- ☐ 1. .open('GET', 'http://monserveur');  
2. .addEventListener('readystatechange', function() {});  
3. .send();
- ☐ 1. .open('GET', 'http://monserveur');  
2. .send();  
3. .addEventListener('readystatechange', function() {});
- ☐ 1. .send();  
2. .addEventListener('readystatechange', function() {});  
3. .open('GET', 'http://monserveur');
- ☐ 1. .addEventListener('readystatechange', function() {});  
2. .open('GET', 'http://monserveur');  
3. .send();

Exercice

Quelles affirmations sont vraies ?

- ☐ Ajouter un header 'Accept : text/html' force le serveur à retourner une réponse au format 'text/html'
- ☐ Le header 'Age: 18' indique l'âge minimum requis pour accéder à une ressource
- ☐ Une requête AJAX permet de communiquer avec un serveur sans recharger la page web
- ☐ Il existe des headers spécifiques aux requêtes et aux réponses
- ☐ Par défaut, une requête lancée avec XMLHttpRequest est synchrone

Exercice



`XmlHttpRequest` permet de suivre l'évolution d'une requête HTTP.

- ☐ Vrai
- ☐ Faux

#### Exercice

Que nous indique un code HTTP 500 ?

- ☐ La requête a été traitée avec succès
- ☐ La réponse à cette requête est ailleurs
- ☐ La syntaxe de la requête est erronée
- ☐ Erreur interne du serveur
- ☐ Je suis une théière

#### Exercice

Quelle méthode permet d'éviter les caractères interdits dans une URL ?

- ☐ `encodeURIComponent()`
- ☐ `encodeURIComponent()`
- ☐ `replaceURI()`
- ☐ `encodeUTF8Component()`

#### Exercice

Si `xhr.readyState === XMLHttpRequest.OPENED` cela signifie...

- ☐ L'objet XHR est créé, mais pas encore initialisé
- ☐ L'objet XHR est initialisé, mais la requête n'est pas envoyée
- ☐ Le transfert de données est en cours
- ☐ Le transfert de données est terminé

#### Exercice

À quoi sert l'objet `FormData` ?

- ☐ À mettre en forme les données de la réponse
- ☐ À faciliter l'envoi d'un formulaire
- ☐ À définir le format de réponse attendu par le client

#### Exercice

Combien d'arguments peut-on passer à la méthode `.send()` de l'objet `XmlHttpRequest` ?

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ Autant que l'on veut

## Exercice

Le *event handler* `onErrorOccured` permet d'écrire un *callback* si une erreur survient avec l'objet `XmlHttpRequest`.

- ☐ Vrai
- ☐ Faux

## B. Exercice : Défi

Maintenant que nous maîtrisons les appels **AJAX**, il est temps de construire notre propre site. Celui-ci aura pour but d'indiquer à un utilisateur les informations d'une ville en fonction de sa recherche. Pour cela, nous allons utiliser l'API du gouvernement : <https://geo.api.gouv.fr/decoupage-administratif/communes#name> à la rubrique **Recherche avancée**.

### Question

[solution n°5 p.26]

À partir du code HTML ci-dessous, construisez le code JavaScript permettant de récupérer le nom d'une ville en fonction du code postal. Le nom des villes sera injecté dans le `Select` sous forme d'option. Pour cela, vous devrez respecter certaines règles.

- Vous n'êtes pas responsable des recherches erronées des utilisateurs (fautes d'orthographe). Retournez un message d'erreur indiquant de recommencer dans le cas où la recherche retourne un objet vide.
- L'utilisateur devra voir :
  - le code postal de la ville,
  - le/les noms de villes recherchées sous forme d'option dans le `select`,
- Le code HTML vous est fourni, vous n'avez pas besoin de le modifier. Il vous faudra utiliser cette structure pour afficher les informations à l'écran.

Toutes les informations nécessaires pour construire la requête vous sont fournies dans la documentation de l'API.

Afin de vous orienter, quelques morceaux de code vous sont proposés : le code HTML et la structure de la fonction stockant une commune dans l'URL.

En revanche, vous devrez établir :

- Une fonction permettant de transformer le texte de la commune en fonction des spécifications.
- Une fonction permettant de créer l'objet à partir des données reçues via l'API.

Fonction pour construire l'URL :

```
1 const apiUrl = 'https://geo.api.gouv.fr/communes?codePostal='
2 const zipcode = document.getElementById("zipcode")
3 const city = document.getElementById("city")
4 const boutonTester = document.getElementById("tester")
5
6 boutonTester.addEventListener("click", (event) => {
7
8   event.preventDefault() // évite que la page ne recharge
9   const code = zipcode.value;
10  const url = apiUrl + code;
11  // votre code ici
12
13 })
```

Code HTML :

```
1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4 <meta charset="utf-8">
5 <title>Geoapi</title>
```

```
6 <linkrel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
7 <style>
8   #container{margin-top: 75px;}
9 </style>
10 </head>
11
12 <body>
13 <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
14 <a class="navbar-brand" href="#">Geoapi</a>
15 </nav>
16 <div class="container" id="container">
17 <div class="starter-template">
18 <form action="" method="post">
19   <div class="form-group">
20     <label for="zipcode">Code Postal</label>
21     <input type="text" name="zipcode" class="form-control" placeholder="Code postal"
22       id="zipcode">
23   <div class="form-group">
24     <label for="city">Ville</label>
25     <select class="form-control" name="city" id="city">
26     </select>
27   </div>
28   <button type="submit" id="tester" class="btn btn-primary">Tester</button>
29 </form>
30 </div>
31 </div>
32 <script src="script.js"></script>
33 </body>
34 </html>
```

Vous utiliserez un environnement de travail repl.it<sup>1</sup>.

### Indice :

Les étapes pour réussir votre code :

- Effectuer une requête sur l'URL.
- Récupérer la réponse dans une variable.
- Boucler sur cette variable pour insérer les résultats dans le Select sous forme d'option.

## Solutions des exercices

---

<sup>1</sup> <https://repl.it/repls/FewNotableCookie>



## p. 6 Solution n°1

```

1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://reqres.in/api/users?page=1');
3
4 xhr.addEventListener('readystatechange', () => {
5     if(xhr.readyState === 4 && xhr.status === 200) {
6         const data = JSON.parse(xhr.response).data;
7         data.forEach(user => {
8             user.name = `${user.first_name} ${user.last_name}`
9         })
10        console.log(data)
11    };
12 });
13
14 xhr.send();

```

## p. 10 Solution n°2

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4     <meta charset="utf-8">
5     <link rel="stylesheet"
6 href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
7 integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk"
8 crossorigin="anonymous">
9     <title>hour</title>
10 </head>
11
12 <body>
13     <div class="container-fluid">
14         <div class="row no-gutters justify-content-between">
15             <div class="col-3">
16                 <form>
17                     <div class="form-group">
18                         <label for="firstName">Nom</label>
19                         <input type="text" name="firstName" class="form-control" id="firstName">
20                     </div>
21                     <div class="form-group">
22                         <label for="lastName">Prénom</label>
23                         <input type="text" name="lastName" class="form-control" id="lastName">
24                     </div>
25                     <div class="form-group">
26                         <label for="grade">Classe</label>
27                         <input type="number" name="grade" class="form-control" id="grade">
28                     </div>
29                     <div class="form-group">
30                         <label for="numberClass">Classe</label>
31                         <input type="number" name="numberClass" class="form-control" id="numberClass">
32                     </div>
33                     <button type="button" class="btn btn-primary"
34 onclick="sendForm()">Enregistrer</button>
35                 </form>
36             </div>
37             <div class="col-8">

```

```

34     <table class="table table-striped text-center">
35         <thead>
36             <tr>
37                 <th>Identifiant</th>
38                 <th>Nom</th>
39                 <th>Prenom</th>
40                 <th>Classe</th>
41                 <th>Numero de classe</th>
42             </tr>
43         </thead>
44         <tbody id="tbody"></tbody>
45     </table>
46 </div>
47 </div>
48 </div>
49 </body>
50
51 <script src="script.js"></script>
52 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
    DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpmoFVY38MBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous">
    </script>
53 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
54 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
    integrity="sha384-OgVRvuATP1z7JjHLku0U7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
    crossorigin="anonymous"></script>
55 </html>

1 const tbody = document.getElementById('tbody');
2 const students = [];
3
4 const createLine = (tbody, student) => {
5     const newRow = tbody.insertRow();
6     let counter = 0
7     for (const key in student) {
8         const newCell = newRow.insertCell(counter);
9         const newText = document.createTextNode(student[key]);
10        newCell.appendChild(newText);
11        counter++
12    }
13 }
14
15 const sendForm = () => {
16     const firstName = document.querySelector('#firstName').value
17     const lastName = document.querySelector('#lastName').value
18     const grade = document.querySelector('#grade').value
19     const numberClass = document.querySelector('#numberClass').value
20
21     const student = {id: '', firstName, lastName, grade, numberClass};
22     const xhr = new XMLHttpRequest();
23     xhr.open('POST', 'https://reqres.in/api/users');
24
25     xhr.addEventListener('readystatechange', () => {
26         if(xhr.readyState === 4) {
27             if (xhr.status === 201) {
28                 const data = JSON.parse(xhr.response);
29                 student.id = data.id;
30                 students.push(student);
31                 createLine(tbody, student);
32             } else {

```

```

33 alert('Erreur')
34 }
35 }
36 });
37
38 xhr.send(student);
39 }
40

```

### p. 14 Solution n°3

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet"
6     href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
7     integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk"
8     crossorigin="anonymous">
9   <title>Connexion</title>
10  <style>
11    .container-fluid {
12      height: 100vh;
13    }
14  </style>
15 </head>
16 <body>
17   <div class="container-fluid">
18     <div class="row no-gutters justify-content-center align-items-center">
19       <div class="col-4">
20         <form>
21           <div class="form-group">
22             <label for="mail">email</label>
23             <input type="text" name="mail" class="form-control" id="mail"
24               value="eve.holt@reqres.in" disabled>
25           </div>
26           <div class="form-group">
27             <label for="psw">Password</label>
28             <input type="password" name="psw" class="form-control" id="psw"
29               value="cityslicka" disabled>
30           </div>
31           <button type="button" class="btn btn-primary" onclick="sendForm(mail.value,
32             psw.value)">Connexion</button>
33         </form>
34       </div>
35     </div>
36   </div>
37 </body>
38 </html>
39
40 <script src="script.js"></script>
41 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
42   Dfxdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous">
43   </script>
44 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
45   integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
46   crossorigin="anonymous"></script>
47 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
48   integrity="sha384-0gVRvuATP1z7JjHLku0U7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JkI"
49   crossorigin="anonymous"></script>
50

```

```

1 let users = []
2
3 // fonction pour récupérer la liste des utilisateurs
4 const getList = (token) => {
5     const newXHR = new XMLHttpRequest();
6     newXHR.open('GET', 'https://reqres.in/api/users?page=1');
7     newXHR.setRequestHeader('Authorization', token);
8     newXHR.addEventListener('readystatechange', () => {
9         if(newXHR.readyState === 4) {
10             if (newXHR.status === 200) {
11                 users = JSON.parse(newXHR.response).data;
12                 console.log(users)
13             } else {
14                 alert('Erreur user')
15             }
16         }
17     });
18     newXHR.send();
19 }
20
21 // Action au click sur le formulaire
22 const sendForm = (email, password) => {
23     setTimeout(() => {
24         const token = 'ffghthfzssz654412dsvcds';
25         getList(token);
26     }, 2000)
27 }

```

### Exercice p. 16 Solution n°4

#### Exercice

Quelle méthode HTTP faut-il utiliser pour récupérer une entité existante depuis le serveur ?

- ☐ HEAD
- ☒ GET
- ☐ OBTAIN
- ☐ PUT
- ☐ POST

#### Exercice

Parmi les éléments ci-dessous, lesquels sont des ordres d'appel valides des méthodes de l'objet XMLHttpRequest ?

- ☒
  1. .open('GET', 'http://monserveur');
  2. .addEventListener('readystatechange', function() {});
  3. .send()
- ☐
  1. .open('GET', 'http://monserveur');
  2. .send()
  3. .addEventListener('readystatechange', function() {});



- ☐ 1. `.send()`  
2. `.addEventListener('readystatechange', function() {});`  
3. `.open('GET', 'http://monserveur');`
- ☒ 1. `.addEventListener('readystatechange', function() {});`  
2. `.open('GET', 'http://monserveur');`  
3. `.send()`

### Exercice

---


Quelles affirmations sont vraies ?

- ☐ Ajouter un header `'Accept : text/html'` force le serveur à retourner une réponse au format `'text/html'`  
*C'est une indication pour le serveur, pas une obligation.*
- ☐ Le header `'Age: 18'` indique l'âge minimum requis pour accéder à une ressource  
*Age: 18 indique la durée en secondes depuis laquelle l'objet a été mis en cache.*
- ☒ Une requête AJAX permet de communiquer avec un serveur sans recharger la page web
- ☒ Il existe des headers spécifiques aux requêtes et aux réponses
- ☐ Par défaut, une requête lancée avec `XmlHttpRequest` est synchrone  
*Elle est **asynchrone**, sauf si on le spécifie explicitement dans la méthode `.open()`.*

### Exercice

---

`XmlHttpRequest` permet de suivre l'évolution d'une requête HTTP.

- ☒ Vrai
- ☐ Faux
-  Grâce au *event handler* `readystatechange`.

### Exercice

---

Que nous indique un code HTTP 500 ?

- ☐ La requête a été traitée avec succès
- ☐ La réponse à cette requête est ailleurs
- ☐ La syntaxe de la requête est erronée
- ☒ Erreur interne du serveur
- ☐ Je suis une théière  
*Ce code existe vraiment, c'est le 418.*

### Exercice

---

Quelle méthode permet d'éviter les caractères interdits dans une URL ?

- ☐ `encodeURIComponent()`
- ☒ `encodeURIComponent()`
- ☐ `replaceURI()`
- ☐ `encodeURIComponent()`

### Exercice

Si `xhr.readyState === XMLHttpRequest.OPENED` cela signifie...

- ☐ L'objet XHR est créé, mais pas encore initialisé
- ☒ L'objet XHR est initialisé, mais la requête n'est pas envoyée
- ☐ Le transfert de données est en cours
- ☐ Le transfert de données est terminé

### Exercice

À quoi sert l'objet `FormData` ?

- ☐ À mettre en forme les données de la réponse
- ☒ À faciliter l'envoi d'un formulaire
- ☐ À définir le format de réponse attendu par le client


### Exercice

Combien d'arguments peut-on passer à la méthode `.send()` de l'objet `XMLHttpRequest` ?

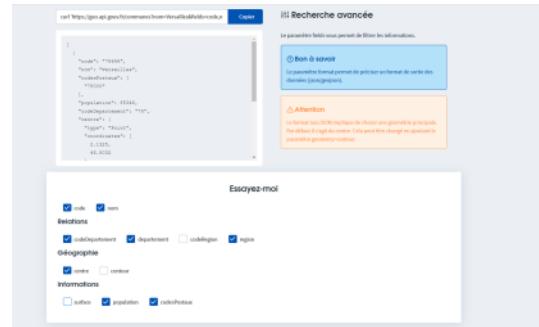
- ☒ 0
- ☒ 1
- ☐ 2
- ☐ Autant que l'on veut

### Exercice

Le *event handler* `onErrorOccured` permet d'écrire un *callback* si une erreur survient avec l'objet `XMLHttpRequest`.

- ☐ Vrai
- ☒ Faux
-  `onErrorOccured` n'existe pas. C'est `onError` qui est implémenté.

En premier lieu, rendons-nous sur la documentation de l'API pour comprendre comment elle fonctionne. On peut voir des cases à cocher, qui sont les différentes informations que nous voulons que l'API nous retourne. Nous remarquons une URL : c'est celle que nous utiliserons pour récupérer les données.



À partir du moment où nous avons tous les éléments en notre possession, nous pouvons à présent écrire le code au complet.

```

1 const apiUrl = 'https://geo.api.gouv.fr/communes?codePostal='
2 const zipcode = document.getElementById("zipcode")
3 const city = document.getElementById("city")
4 const boutonTester = document.getElementById("tester")
5
6 boutonTester.addEventListener("click", (event)=>{
7
8     event.preventDefault() // évite que la page ne recharge
9
10    const code = zipcode.value;
11    const url = apiUrl+code;
12
13    let xhr = new XMLHttpRequest();
14    xhr.open("GET", url);
15    xhr.onload = () => {
16
17        if (xhr.status === 200) {
18
19            const result = JSON.parse(xhr.response);
20
21            if(result.length){
22
23                result.forEach(function(value, key){
24
25                    let myOption = document.createElement("option");
26                    myOption.text = value.nom;
27                    myOption.value = value.nom;
28                    city.appendChild(myOption);
29                });
30            }
31        }else{
32            console.log('L\'appel API a échoué');
33        }
34    };
35    xhr.send();
36
37 })

```