**Étape 1 — Premiers pas avec la syntaxe de l’API Fetch**

Pour utiliser l’API Fetch, appelez la méthode fetch, qui accepte l’URL de l’API comme paramètre :

fetch(url)

Copy

Après la méthode fetch(), ajoutez la méthode de promesse then() :

.then(function() {

})

Copy

La méthode fetch() retourne une promesse. Si la promesse renvoyée est resolve, cela signifie que la fonction dans la méthode then() est bien exécutée. Cette fonction contient le code qui permet de traiter les données reçues à partir de l’API.

Sous la méthode then(), ajoutez la méthode catch() :

.catch(function() {

});

Copy

Il se peut que l’API que vous appelez en utilisant fetch() soit défaillante ou que d’autres erreurs se produisent. Le cas échéant, le système renverra la promesse reject. Vous pouvez utiliser la méthode cath pour gérer reject. Le code dans catch() sera exécuté dans le cas où une erreur se produit lors de l’appel de l’API de votre choix.

Pour résumer, l’utilisation de l’API Fetch ressemblera à l’exemple suivant :

fetch(url)

.then(function() {

})

.catch(function() {

});

Copy

En ayant une compréhension de la syntaxe qui permet d’utiliser l’API Fetch, vous pouvez désormais passer à l’utilisation de fetch() sur une API réelle.

**Étape 2 — Utilisation de Fetch pour obtenir des données à partir d’une API**

Les échantillons de code suivants seront basés sur l’[API Random User](https://randomuser.me/). En utilisant l’API, vous obtiendrez dix utilisateurs que vous pourrez afficher sur la page en utilisant Vanilla JavaScript.

L’idée est de récupérer toutes les données de l’API Random User et de les afficher dans les éléments de la liste de l’auteur. Commencez par créer un fichier HTML, ajoutez un titre et une liste non ordonnée avec l’id des authors :

<h1>Authors</h1>

<ul id="authors"></ul>

Copy

Ajoutez maintenant les balises script au bas de votre fichier HTML et utilisez un sélecteur DOM pour récupérer l’ul. Utilisez l’argument getElementById avec authors. N’oubliez pas, authors est l’id de l’ul précédemment créé :

<script>

const ul = document.getElementById('authors');

</script>

Copy

Créez une variable constante nommée url qui contiendra l’URL de l’API qui renverra dix utilisateurs aléatoires :

const url = 'https://randomuser.me/api/?results=10';

Copy

Une fois que ul et url sont créés, il est temps pour vous de créer les fonctions qui serviront à créer les éléments de la liste. Créez une fonction appelée createNode qui prend un paramètre appelé element :

function createNode(element) {

}

Copy

Plus tard, lorsque vous appellerez createNode, vous devrez transmettre le nom d’un élément HTML réel à créer.

Dans la fonction, ajoutez une instruction return qui renvoie element en utilisant document.createElement() :

function createNode(element) {

return document.createElement(element);

}

Copy

En outre, vous devrez créer une fonction appelée append qui intègre deux paramètres : parent et el :

function append(parent, el) {

}

Copy

Cette fonction ajoutera el au parent en utilisant document.createElement :

function append(parent, el) {

return parent.appendChild(el);

}

Copy

createNode et append sont tous deux prêts à être utilisés. Maintenant, en utilisant l’API Fetch, appelez l’API Random User en utilisant fetch() avec l’argument url :

fetch(url)

Copy

fetch(url)

.then(function(data) {

})

})

.catch(function(error) {

});

Copy

Dans le code ci-dessus, vous appelez l’API Fetch et passez l’URL à l’API Random User. Ensuite, vous recevrez une réponse. Cependant, la réponse que vous obtenez n’est pas JSON, mais plutôt un objet avec une série de méthodes utilisables en fonction de ce que vous voulez faire avec les informations. Pour convertir l’objet renvoyé en JSON, utilisez la méthode json().

Ajoutez la méthode then() qui contiendra une fonction avec un paramètre appelé resp :

fetch(url)

.then((resp) => )

Copy

Le paramètre resp prend la valeur de l’objet renvoyé de fetch(url). Utilisez la méthode json() pour convertir resp en données JSON :

fetch(url)

.then((resp) => resp.json())

Copy

Il vous reste encore à traiter les données JSON. Ajoutez une autre instruction then() avec une fonction dont l’argument s’appelle data :

.then(function(data) {

})

})

Copy

Dans cette fonction, créez une variable appelée authors, qui est définie comme égale à data.results :

.then(function(data) {

let authors = data.results;

Copy

Pour chaque auteur dans authors, il vous faudra créer un élément de liste qui affiche leur photographie et leur nom respectifs. La méthode map() est idéale pour faire cela :

let authors = data.results;

return authors.map(function(author) {

})

Copy

Dans votre fonction map, créez une variable appelée li qui sera définie comme égale à createNode avec li (l’élément HTML) comme argument :

return authors.map(function(author) {

let li = createNode('li');

})

Copy

Répétez cette manœuvre pour créer un élément span et un élément img :

let li = createNode('li');

let img = createNode('img');

let span = createNode('span');

Copy

L’API propose un nom pour l’auteur et une image qui accompagne le nom. Définissez l’img.src sur l’image de l’auteur :

let img = createNode('img');

let span = createNode('span');

img.src = author.picture.medium;

Copy

L’élément span doit contenir le prénom et le nom de famille de l’auteur. Pour cela, utilisez la propriété innerHTML et l’interpolation des chaînes de caractères :

img.src = author.picture.medium;

span.innerHTML = `${author.name.first} ${author.name.last}`;

Copy

Une fois l’image et l’élément de liste créés avec l’élément span, vous pouvez utiliser la fonction append précédemment créée pour afficher ces éléments sur la page :

append(li, img);

append(li, span);

append(ul, li);

Copy

Une fois les deux fonctions then() terminées, vous pouvez maintenant ajouter la fonction catch(). Cette fonction enregistrera l’erreur potentielle sur la console :

.catch(function(error) {

console.log(error);

});

Copy

Voici le code complet de la requête que vous avez créée :

function createNode(element) {

return document.createElement(element);

}

function append(parent, el) {

return parent.appendChild(el);

}

const ul = document.getElementById('authors');

const url = 'https://randomuser.me/api/?results=10';

fetch(url)

.then((resp) => resp.json())

.then(function(data) {

let authors = data.results;

return authors.map(function(author) {

let li = createNode('li');

let img = createNode('img');

let span = createNode('span');

img.src = author.picture.medium;

span.innerHTML = `${author.name.first} ${author.name.last}`;

append(li, img);

append(li, span);

append(ul, li);

})

})

.catch(function(error) {

console.log(error);

});

Copy

Vous avez réussi à réaliser une requête GET en utilisant l’API Random User et l’API Fetch. Au cours de la prochaine étape, vous allez apprendre à réaliser des requêtes POST.

**Étape 3 — Traitement des requêtes POST**

Fetch utilise par défaut les requêtes GET, mais vous pouvez utiliser tous les autres types de requêtes, changer les en-têtes et envoyer des données. Pour ce faire, vous devez configurer votre objet et le passer comme le deuxième argument de la fonction fetch.

Avant de créer une requête POST, créez les données que vous souhaitez envoyer à l’API. Il s’agira d’un objet appelé data avec le name clé et la valeur Sammy (ou votre nom) :

const url = 'https://randomuser.me/api';

let data = {

name: 'Sammy'

}

Copy

Veillez à bien inclure une variable constante qui contient le lien vers l’API Random User.

Comme il s’agit d’une requête POST, vous devrez l’indiquer de manière explicite. Créez un objet appelé fetchData :

let fetchData = {

}

Copy

Cet objet doit inclure trois clés : method, body et headers. La clé method doit avoir la valeur 'POST'. body doit être défini comme égal à l’objet data qui vient d’être créé. headers devrait avoir la valeur des new Headers() :

let fetchData = {

method: 'POST',

body: data,

headers: new Headers()

}

Copy

L’interface Headers est une propriété de l’API Fetch qui vous permet d’effectuer diverses actions sur les en-têtes des requêtes et des réponses HTTP. Si vous souhaitez en savoir plus à ce sujet, vous pourrez trouver de plus amples informations dans l’article suivant : [Comment définir les routes et les méthodes de requête HTTP dans Express](https://www.digitalocean.com/community/tutorials/nodejs-express-routing).

Une fois ce code en place, vous pouvez réaliser la requête POST en utilisant l’API Fetch. Vous allez inclure url et fetchData comme arguments pour votre requête POST fetch :

fetch(url, fetchData)

Copy

La fonction then() comprendra un code qui gère la réponse reçue du serveur API Random User :

fetch(url, fetchData)

.then(function() {

// Handle response you get from the server

});

Copy

Vous disposez également d’une autre option pour créer un objet et utiliser la fonction fetch(). Au lieu de créer un objet comme fetchData, vous pouvez utiliser le constructeur de requêtes pour créer votre objet de requête. Pour ce faire, créez une variable appelée request :

const url = 'https://randomuser.me/api';

let data = {

name: 'Sara'

}

var request =

Copy

La variable request doit être configurée comme new Request. La construction New Request prend deux arguments : l’url de l’API url (url) et un objet. L’objet doit également inclure les clés method, body et headers tout comme fetchData :

var request = new Request(url, {

method: 'POST',

body: data,

headers: new Headers()

});

Copy

Maintenant, vous pouvez utiliser request comme le seul argument de fetch() car il inclut également l’url de l’API :

fetch(request)

.then(function() {

// Handle response we get from the API

})

Copy

L’ensemble de votre code ressemblera à ce qui suit :

const url = 'https://randomuser.me/api';

let data = {

name: 'Sara'

}

var request = new Request(url, {

method: 'POST',

body: data,

headers: new Headers()

});

fetch(request)

.then(function() {

// Handle response we get from the API

})

Copy

Vous connaissez maintenant deux méthodes qui vous permettront de créer et d’exécuter des requêtes POST avec l’API Fetch.