

# Présentation des API

# Table des matières

<b>I. Qu'est-ce qu'une API ?</b>	<b>3</b>
<b>II. Exercice : Appliquer la notion</b>	<b>5</b>
<b>III. API basées sur le protocole HTTP : REST</b>	<b>5</b>
<b>IV. Exercice : Appliquer la notion</b>	<b>9</b>
<b>V. API à contrat strict : le protocole SOAP</b>	<b>9</b>
<b>VI. Exercice : Appliquer la notion</b>	<b>15</b>
<b>VII. Autres façons de procéder avec fetch()</b>	<b>15</b>
<b>VIII. Essentiel</b>	<b>17</b>
<b>IX. Auto-évaluation</b>	<b>17</b>
A. Exercice .....	17
B. Test .....	17
<b>Solutions des exercices</b>	<b>19</b>

## I. Qu'est-ce qu'une API ?

**Durée :** 1 h

**Environnement de travail :** Windows / (Mac OSX)

**Pré-requis :**

- Bases en HTTP (GET, POST, PUT, DELETE)
- Bases en XML
- Bases en Javascript

### Contexte

Les API (Application Programming Interface) ou encore interfaces de programmation applicatives sont énormément utilisées dans les applications modernes qui sont ultra connectées à différents services.

Par exemple, qui peut imaginer aujourd'hui une application qui ne dispose pas d'un système de connexion à l'aide d'un réseau social, ou encore pour un site e-commerce, de certaines méthodes de paiement telles que Stripe ou Paypal ?

Mais comment allons-nous pouvoir communiquer avec ces différents services développés par d'autres ? C'est très simple, nous allons utiliser les API fournies par ces services.

Dans ce cours nous allons décrire les différents modèles d'API et leurs méthodes d'accès.

API ouvertes, fermées, SOAP ou encore REST, etc. À la fin de ce cours, tous ces termes n'auront plus aucun secret pour vous.

### Objectifs

- Comprendre la notion d'API
- Les API ouvertes
- Les API fermées

### Contexte

Les API sont des interfaces de communication entre deux systèmes ! En ce sens, si vous avez déjà développé et partagé une librairie dans votre langage favori, vous avez déjà développé une API.

En effet, les fonctions que vous avez mises à disposition des développeurs et développeuses ne sont-elles pas une « *interface de communication* » vers des parties plus privées de votre projet ?

Vous l'aurez compris, il existe des API pour tout : pour le paiement, pour les accès aux fonctionnalités des réseaux sociaux, pour afficher une fenêtre de support sur une page d'accueil, ou encore pour accéder à des données en JSON ou XML.

Dans cette multitude d'API disponibles, certaines seront considérées ouvertes et d'autres fermées.

Ce qui différencie une API ouverte d'une API fermée, c'est que pour accéder à une API fermée nous allons devoir utiliser une méthode d'authentification qui permette à cette API d'évaluer si nous sommes autorisés à accéder à ces fonctionnalités.

Par exemple, imaginons que vous souhaitiez récupérer une liste de tweets dont le sujet est les API.

Le réseau social Twitter fournit une API permettant de récupérer la liste des tweets en fonction d'un sujet, mais elle vous demande de créer un compte « *Développeur* »<sup>1</sup> et de vous identifier avant de pouvoir accéder aux données: c'est ce que l'on appelle une API fermée.

### Méthode

Chacune des API disponibles dispose de ses propres règles qui seront documentées.

Pourtant, 2 modèles d'API sont très populaires et utilisés pour partager des fonctionnalités et des données : les API **REST** et les API **SOAP** que nous aborderons en détail.

### Exemple

Pour pouvoir communiquer avec une API, nous devons donc maîtriser un langage de programmation et consulter sa documentation. Ici, nous allons consulter une API permettant de lister des blagues<sup>2</sup> (« jokes » en anglais).

La documentation<sup>3</sup> nous informe de l'URL à utiliser pour se connecter à cette API et des points d'entrées à utiliser:

- Accéder à jod<sup>4</sup>, partie de l'API gratuite/ouverte, nous retournera la blague du jour et toute une série d'informations sur cette blague au format JSON/XML ou JSONP.
- Accéder à jokes<sup>5</sup>, partie payante de l'API nous permettrait d'accéder à tout un tas d'option comme le choix d'une blague aléatoire, la création ou la suppression d'une blague.

Nous avons donc ici une partie de l'API qui est ouverte (« jod »), et une partie de l'API qui est fermée (« jokes »).

Il s'agit donc en Javascript (pour la partie ouverte, ou encore « publique ») de récupérer du JSON à partir d'une URL. Voici un exemple de code fonctionnel de communication avec cette API :

```
1  const request = new XMLHttpRequest();
2  request.open('GET', 'https://api.jokes.one/jod/', true);
3
4  request.onload = function() {
5    if (this.status >= 200 && this.status < 400) {
6      // Accès à l'API réussi !
7      let = JSON.parse(this.response);
8      console.log(data) ;
9    } else {
10     // Erreur retournée par le serveur
11    }
```

À l'aide d'un navigateur web et en intégrant ce code Javascript dans une page HTML vous devriez obtenir un résultat équivalent à celui-ci en accédant à la console de développement.

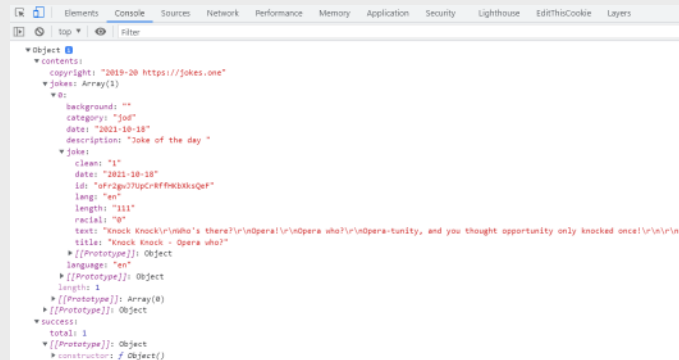
1 <https://developer.twitter.com/en/docs/twitter-api/getting-started/guide>

2 <https://jokes.one/>

3 <https://jokes.one/api/joke/>

4 <https://api.jokes.one/jod/>

5 <https://cat-fact.herokuapp.com/users>



### Syntaxe À retenir

Les API sont des interfaces vous permettant d'accéder à des données et des fonctionnalités fournies par d'autres développeurs et développeuses.

Elles sont divisées entre les API ouvertes (ou publiques) qui sont directement accessibles et des API fermées qui nécessitent une méthode d'authentification.

En quelques lignes de Javascript il est possible de se connecter à une API, mais il faudra consulter la documentation de chaque API pour savoir comment y accéder.

### Complément

- Le client HTTP Postman<sup>1</sup>
- Liste d'APIs publiques REST<sup>(EN)2</sup>
- Liste d'APIs publiques SOAP<sup>(EN)3</sup>

## II. Exercice : Appliquer la notion

### Question

[solution n°1 p.21]

Vous allez utiliser une API ouverte pour récupérer la liste de tous les personnages de la série « *Game of Thrones* ». Pour cela, prenez le temps de consulter la documentation<sup>4</sup> de l'API ouverte « *Ice and Fire* » et écrivez un script Javascript capable de récupérer la liste des 20 premiers personnages.

**Quel est le nom du personnage avec l'identifiant n°12 ?**

#### Indice :

Par défaut, l'API retourne seulement 10 éléments et il vous en faut au moins 12, il est peut-être possible de changer la taille de la pagination.

## III. API basées sur le protocole HTTP : REST

1 <https://www.postman.com/downloads/>

2 <https://documenter.getpostman.com/view/8854915/Szf7znEe?version=latest>

3 <https://documenter.getpostman.com/view/8854915/Szf26WHn?version=latest>

4 <https://anapiofireandice.com/Documentation>

## Objectifs

- Rappels sur le protocole HTTP
- Comprendre la méthode REST
- Savoir communiquer avec une API REST

### Contexte

Aujourd'hui, l'essentiel des API suivent non pas un protocole mais plutôt une méthode appelée REST.

REST veut dire « *Representational State Transfert* », c'est une architecture qui définit un ensemble de contraintes pour la conception de services web.

Les API REST sont basées sur le protocole HTTP et sont définies par :

- Une **URI** que l'on appelle également « *point d'entrée* » de l'API,
- Un ensemble de méthodes HTTP : GET, POST, PUT, PATCH et DELETE,
- Un format de retour structuré : qui peut être du XML, ou du JSON.

### Relation entre méthode HTTP et fonctionnalités de l'API

Méthode HTTP	Fonctionnalité recommandée
GET	Récupération d'information(s), aucune écriture de nouvelles données dans l'API. <i>Ex : récupérer un utilisateur à l'aide de son identifiant.</i>
POST	Insertion de nouvelle(s) information(s) dans l'API. <i>Ex : création d'un nouvel utilisateur.</i>
PUT, PATCH	Modification d'information(s) dans l'API. <i>Ex : mise à jour d'un utilisateur.</i>
DELETE	Suppression d'information(s) dans l'API. <i>Ex : suppression d'un utilisateur.</i>

Dans l'architecture REST, les éléments que nous manipulons sont appelés des **ressources**.

Ainsi, si l'on réalise une API capable de manipuler des produits, nous dirons que les produits sont des ressources manipulées par l'API.

## Le modèle de maturité de Richardson

Développé en 2010, ce modèle permet d'évaluer la qualité de l'API en fonction de son respect de l'architecture définie. Sa qualité se définit en « *niveau* » de maturité à atteindre.

Niveau de maturité	Contrainte validée par l'API
Niveau 0 : HTTP	L'API doit utiliser le protocole de communication <b>HTTP</b> pour ses interactions avec les clients de l'API.
Niveau 1 : Ressources	L'API doit manipuler des <b>ressources</b> . Concrètement, cela veut dire qu'à chaque ressource manipulée correspond un format identifié et associé à une URL.
Niveau 2 : Verbes HTTP	Pour chaque action disponible dans l'API sont associées les méthodes HTTP adéquates (telles que définies dans le tableau précédent).
Niveau 3 : Hypermedia	Pour chaque action disponible de l'API, elle retourne un accès vers toutes les autres actions disponibles. On parle alors d'API « <i>auto documentée</i> » ou « <i>auto découvrable</i> ». <i>Ex : après création d'un utilisateur, on reçoit en réponse le lien vers « voir l'utilisateur », « éditer l'utilisateur » et « supprimer l'utilisateur ».</i>

L'essentiel des API REST disponibles aujourd'hui sont en réalité des API de niveau 2.

### Méthode

Pour utiliser une API REST, nous avons donc besoin de manipuler des ressources en passant par le protocole HTTP en utilisant un point d'entrée (l'URI) et la méthode HTTP adaptée.

Nous devons donc développer un client HTTP en Javascript capable de recevoir et d'envoyer de l'information à un serveur au travers de l'URI de l'API.

C'est une fonctionnalité qui est native au langage et qui passe par un objet dont nous allons détailler le fonctionnement : **XMLHttpRequest**.

Une fois l'objet instancié, deux fonctions sont importantes pour notre besoin : `open()` et `send()`.

La fonction `open()` permet de définir la méthode HTTP et l'URI, tandis que la méthode `send()` permet d'exécuter la « *requête HTTP* » (de la même façon que lorsque l'on entre une URL dans un navigateur web).

Par exemple, considérez le code Javascript suivant :

```
1 const xmlHttp = new XMLHttpRequest();
2 xmlHttp.open('GET', 'https://www.studi.fr/', false);
3 xmlHttp.send();
4 console.log(xmlHttp.responseText);
```

Ce code va permettre de récupérer le contenu de la page d'accueil du site de Studi.

### La fonction `open()` a 3 arguments :

- La méthode HTTP : GET, POST, PUT, PATCH, DELETE,
- L'URI (ou encore point d'entrée de l'API dans le contexte),
- Un paramètre booléen qui permet de définir si nous souhaitons une action asynchrone ou non.

Si nous souhaitons effectuer plutôt des appels asynchrones (plus performant, recommandé), voici l'exemple précédent adapté :

```
1 const request = new XMLHttpRequest();
2 request.open('GET', 'https://www.studi.fr/', true);
3
4 request.onload = function() {
5     // La requête s'est correctement effectuée
6     if (this.status >= 200 && this.status < 400) {
7         console.log(this.response.responseText)
8     }
9 }
10
11 request.onerror = function() {
12     // Le serveur a retourné une erreur
13 }
14 request.send();
```

**La fonction send()** permet d'exécuter la requête HTTP et d'envoyer de la donnée si nécessaire :

```
1 const request = new XMLHttpRequest();
2 request.open('POST', 'https://postman-echo.com/post');
3 request.setRequestHeader('Content-type', 'application/json');
4 request.send('Bonjour Studi, comment vas-tu ?');
5
6 request.onload = function() {
7     response_from_api = JSON.parse(request.responseText, true)
8     response = document.getElementById('response')
9     console.log(response_from_api.data)
10    response.innerHTML = response_from_api.data
```

### Exemple

Voici un code fonctionnel en Javascript, qui va récupérer à chaque appel une blague différente :

```
1 const request = new XMLHttpRequest();
2 request.open('GET', 'https://official-joke-api.appspot.com/random_joke');
3
4 request.onload = function() {
5     response_from_server = JSON.parse(request.responseText);
6     question = document.getElementById('question');
7     punchline = document.getElementById('punchline');
8
9     // Charger le contenu
10    question.innerHTML = response_from_server.setup;
11    punchline.innerHTML = response_from_server.punchline;
12 }
13 request.send();
```

**Attention !** L'API n'est plus disponible directement mais via le git suivant : [GitHub - 15Dkatz/official\\_joke\\_api](https://github.com/15Dkatz/official_joke_api): Official Joke API!<sup>1</sup>

### Syntaxe À retenir

Une API REST définit un ensemble d'opérations disponibles sur des ressources.

En exécutant des requêtes HTTP sur le point d'entrée de l'API, à l'aide de la bonne méthode HTTP et des paramètres (s'il y en a) nous communiquons au serveur nos intentions.

<sup>1</sup> [https://github.com/15Dkatz/official\\_joke\\_api](https://github.com/15Dkatz/official_joke_api)



En JavaScript, l'objet **XMLHttpRequest** nous permet d'exécuter des requêtes HTTP.

Deux fonctions sont à maîtriser :

- La fonction `open()`
- La fonction `send()`

Elles permettent respectivement de définir l'URI, la méthode HTTP et de provoquer l'envoi de la requête HTTP.

#### Complément

- Modèle de Richardson<sup>(EN)<sup>1</sup></sup>
- L'objet XMLHttpRequest<sup>2</sup>

## IV. Exercice : Appliquer la notion

### Question

[solution n°2 p.21]

Space X est une société de lancement de fusées et satellites dans l'espace, fondée par Elon Musk.

À partir de la documentation et à l'aide de vos nouvelles connaissances, quelle est la date du premier lancement de l'entreprise.

### Indice :

Un « *lancement* » se traduit par « *launch* » en anglais, nous souhaitons ici récupérer une information sur des lancements déjà effectués dans le passé.

## V. API à contrat strict : le protocole SOAP

### Objectifs

- Comprendre la notion de contrat (pour les API)
- Savoir expliquer comment fonctionne le protocole SOAP
- Savoir communiquer avec une API SOAP

#### Contexte

Jusque-là, nous avons appris que les API étaient de toute sorte et que la seule façon pour arriver à les manipuler était de consulter la documentation. Avec la multiplication des API, l'entreprise Microsoft, en collaboration avec le W3C (organisme qui décrit les standards du web), a créé le protocole SOAP comme un moyen de communication entre applications web utilisant HTTP et le langage XML.

Le but est d'avoir une certaine interopérabilité et facilité dans l'utilisation des API. Une fois le protocole maîtrisé, il est plus facile d'intégrer de nouvelles API puisqu'elles fonctionnent toutes selon les mêmes règles !

1 <https://martinfowler.com/articles/richardsonMaturityModel.html>

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

Concrètement, comment ça fonctionne ? Le fournisseur de l'API va fournir un fichier XML (on l'appelle WSDL dans ce contexte) avec les informations suivantes :

- Le **protocole de communication** à utiliser,
- Le **format des messages** pour communiquer avec son API,
- Les **méthodes utilisables** par les clients (c'est à dire, vous),
- La **localisation du service** (l'URL).

*Vous n'avez pas tellement à vous préoccuper de ce fichier car il est fourni par le fournisseur de l'API.*

Si on résume, un serveur SOAP utilise donc un langage particulier (le Web Services Description Language ou encore **WSDL**) pour définir les actions possibles aux utilisateurs et utilisatrices de l'API.

Pour communiquer avec une API SOAP, il faudra donc utiliser le langage XML et envoyer 2 types d'informations :

- Les en-têtes (format, méthode à utiliser),
- Ce que l'on appelle l'enveloppe SOAP qui contient vos données, si vous devez en envoyer au serveur SOAP.

### Méthode

Dans la pratique, pour chaque API SOAP nous allons consulter le fichier WSDL qui contient donc les informations nécessaires pour effectuer une requête.

Prenons une API SOAP ouverte<sup>1</sup> qui permet d'effectuer une conversion de degré Celsius en degré Fahrenheit et vice-versa. Voici le fichier qui décrit l'API (le fichier WSDL) :

```
1 <wsdl:definitions xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="https://www.w3schools.com/xml/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="https://www.w3schools.com/xml/">
2 <wsdl:types>
3 <s:schema elementFormDefault="qualified" targetNamespace="https://www.w3schools.com/xml/">
4 <s:element name="FahrenheitToCelsius">
5 <s:complexType>
6 <s:sequence>
7 <s:element minOccurs="0" maxOccurs="1" name="Fahrenheit" type="s:string"/>
8 </s:sequence>
9 </s:complexType>
10 </s:element>
11 <s:element name="FahrenheitToCelsiusResponse">
12 <s:complexType>
13 <s:sequence>
14 <s:element minOccurs="0" maxOccurs="1" name="FahrenheitToCelsiusResult" type="s:string"/>
15 </s:sequence>
16 </s:complexType>
17 </s:element>
18 <s:element name="CelsiusToFahrenheit">
19 <s:complexType>
20 <s:sequence>
21 <s:element minOccurs="0" maxOccurs="1" name="Celsius" type="s:string"/>
22 </s:sequence>
23 </s:complexType>
24 </s:element>
25 <s:element name="CelsiusToFahrenheitResponse">
26 <s:complexType>
```

<sup>1</sup> <https://www.w3schools.com/xml/tempconvert.asmx>

```

27 <s:sequence>
28 <s:element minOccurs="0" maxOccurs="1" name="CelsiusToFahrenheitResult" type="s:string"/>
29 </s:sequence>
30 </s:complexType>
31 </s:element>
32 <s:element name="string" nillable="true" type="s:string"/>
33 </s:schema>
34 </wsdl:types>
35 <wsdl:message name="FahrenheitToCelsiusSoapIn">
36 <wsdl:part name="parameters" element="tns:FahrenheitToCelsius"/>
37 </wsdl:message>
38 <wsdl:message name="FahrenheitToCelsiusSoapOut">
39 <wsdl:part name="parameters" element="tns:FahrenheitToCelsiusResponse"/>
40 </wsdl:message>
41 <wsdl:message name="CelsiusToFahrenheitSoapIn">
42 <wsdl:part name="parameters" element="tns:CelsiusToFahrenheit"/>
43 </wsdl:message>
44 <wsdl:message name="CelsiusToFahrenheitSoapOut">
45 <wsdl:part name="parameters" element="tns:CelsiusToFahrenheitResponse"/>
46 </wsdl:message>
47 <wsdl:message name="FahrenheitToCelsiusHttpPostIn">
48 <wsdl:part name="Fahrenheit" type="s:string"/>
49 </wsdl:message>
50 <wsdl:message name="FahrenheitToCelsiusHttpPostOut">
51 <wsdl:part name="Body" element="tns:string"/>
52 </wsdl:message>
53 <wsdl:message name="CelsiusToFahrenheitHttpPostIn">
54 <wsdl:part name="Celsius" type="s:string"/>
55 </wsdl:message>
56 <wsdl:message name="CelsiusToFahrenheitHttpPostOut">
57 <wsdl:part name="Body" element="tns:string"/>
58 </wsdl:message>
59 <wsdl:portType name="TempConvertSoap">
60 <wsdl:operation name="FahrenheitToCelsius">
61 <wsdl:input message="tns:FahrenheitToCelsiusSoapIn"/>
62 <wsdl:output message="tns:FahrenheitToCelsiusSoapOut"/>
63 </wsdl:operation>
64 <wsdl:operation name="CelsiusToFahrenheit">
65 <wsdl:input message="tns:CelsiusToFahrenheitSoapIn"/>
66 <wsdl:output message="tns:CelsiusToFahrenheitSoapOut"/>
67 </wsdl:operation>
68 </wsdl:portType>
69 <wsdl:portType name="TempConvertHttpPost">
70 <wsdl:operation name="FahrenheitToCelsius">
71 <wsdl:input message="tns:FahrenheitToCelsiusHttpPostIn"/>
72 <wsdl:output message="tns:FahrenheitToCelsiusHttpPostOut"/>
73 </wsdl:operation>
74 <wsdl:operation name="CelsiusToFahrenheit">
75 <wsdl:input message="tns:CelsiusToFahrenheitHttpPostIn"/>
76 <wsdl:output message="tns:CelsiusToFahrenheitHttpPostOut"/>
77 </wsdl:operation>
78 </wsdl:portType>
79 <wsdl:binding name="TempConvertSoap" type="tns:TempConvertSoap">
80 <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
81 <wsdl:operation name="FahrenheitToCelsius">
82 <soap:operation soapAction="https://www.w3schools.com/xml/FahrenheitToCelsius"
    style="document"/>

```

```

83 <wsdl:input>
84 <soap:body use="literal"/>
85 </wsdl:input>
86 <wsdl:output>
87 <soap:body use="literal"/>
88 </wsdl:output>
89 </wsdl:operation>
90 <wsdl:operation name="CelsiusToFahrenheit">
91 <soap:operation soapAction="https://www.w3schools.com/xml/CelsiusToFahrenheit"
  style="document"/>
92 <wsdl:input>
93 <soap:body use="literal"/>
94 </wsdl:input>
95 <wsdl:output>
96 <soap:body use="literal"/>
97 </wsdl:output>
98 </wsdl:operation>
99 </wsdl:binding>
100 <wsdl:binding name="TempConvertSoap12" type="tns:TempConvertSoap">
101 <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
102 <wsdl:operation name="FahrenheitToCelsius">
103 <soap12:operation soapAction="https://www.w3schools.com/xml/FahrenheitToCelsius"
  style="document"/>
104 <wsdl:input>
105 <soap12:body use="literal"/>
106 </wsdl:input>
107 <wsdl:output>
108 <soap12:body use="literal"/>
109 </wsdl:output>
110 </wsdl:operation>
111 <wsdl:operation name="CelsiusToFahrenheit">
112 <soap12:operation soapAction="https://www.w3schools.com/xml/CelsiusToFahrenheit"
  style="document"/>
113 <wsdl:input>
114 <soap12:body use="literal"/>
115 </wsdl:input>
116 <wsdl:output>
117 <soap12:body use="literal"/>
118 </wsdl:output>
119 </wsdl:operation>
120 </wsdl:binding>
121 <wsdl:binding name="TempConvertHttpPost" type="tns:TempConvertHttpPost">
122 <http:binding verb="POST"/>
123 <wsdl:operation name="FahrenheitToCelsius">
124 <http:operation location="/FahrenheitToCelsius"/>
125 <wsdl:input>
126 <mime:content type="application/x-www-form-urlencoded"/>
127 </wsdl:input>
128 <wsdl:output>
129 <mime:mimeXml part="Body"/>
130 </wsdl:output>
131 </wsdl:operation>
132 <wsdl:operation name="CelsiusToFahrenheit">
133 <http:operation location="/CelsiusToFahrenheit"/>
134 <wsdl:input>
135 <mime:content type="application/x-www-form-urlencoded"/>
136 </wsdl:input>
137 <wsdl:output>

```

```

138 <mime:mimeXml part="Body"/>
139 </wsdl:output>
140 </wsdl:operation>
141 </wsdl:binding>
142 <wsdl:service name="TempConvert">
143 <wsdl:port name="TempConvertSoap" binding="tns:TempConvertSoap">
144 <soap:address location="http://www.w3schools.com/xml/tempconvert.asmx"/>
145 </wsdl:port>
146 <wsdl:port name="TempConvertSoap12" binding="tns:TempConvertSoap12">
147 <soap12:address location="http://www.w3schools.com/xml/tempconvert.asmx"/>
148 </wsdl:port>
149 <wsdl:port name="TempConvertHttpPost" binding="tns:TempConvertHttpPost">
150 <http:address location="http://www.w3schools.com/xml/tempconvert.asmx"/>
151 </wsdl:port>
152 </wsdl:service>
153 </wsdl:definitions>

```

Ce fichier peut sembler incompréhensible au premier abord et c'est normal ! En tant que développeur ou développeuse web, il est très rare d'obtenir de la documentation au format XML.

Ce qui nous intéresse ici, ce sont 3 informations nécessaires pour appeler l'API :

- Le nom de l'opération (ou fonctionnalité, ou point d'entrée),
- Les paramètres nécessaires pour utiliser cette opération et leur type,
- Le point d'entrée de l'API à utiliser pour cette opération.

### Nom de l'opération

Les opérations sont listées en tant qu'élément XML « *wsdl:operation* » et ont un attribut *name*. Ici, il n'y a que 2 opérations :

- FahrenheitToCelsius
- CelsiusToFahrenheit

### Les paramètres nécessaires à l'opération

Nous retrouverons les paramètres nécessaires pour chacune des opérations dans la section « *wsdl:types* » du fichier.

Pour l'élément "CelsiusToFahrenheit" voici la description du type :

```

<s:element name="CelsiusToFahrenheit">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="Celsius" type="s:string"/>
</s:sequence>
</s:complexType>
</s:element>

```

En Français, cela veut dire que cette opération nécessite un élément **Celsius** (et un seul) qui acceptera une valeur de type « *string* ».

### Le point d'entrée de l'API

Pour cela nous regarderons les nœuds XML « *wsdl:definitions* » et « *wsdl:address* » et notamment la propriété « *location* ».

Nous obtenons ici l'URL<sup>1</sup> et 2 versions de SOAP supportées :

SOAP (SOAP 1.0) et SOAP 12 (SOAP 1.2)

*Nous n'aborderons pas ici les différences entre ces deux versions.*

À partir de toutes ces informations, nous pouvons donc **décrire** la requête à effectuer sur ce serveur.

Ci-dessous un exemple de code fonctionnel en Javascript.

#### Exemple

```

1 <html>
2 <head>
3   <title>Simple client SOAP en JavaScript</title>
4   <script type="text/javascript">
5     function soap() {
6       const xmlhttp = new XMLHttpRequest();
7       xmlhttp.open('POST', 'https://www.w3schools.com/xml/tempconvert.asmx', true);
8
9       let form_celsius = document.getElementById('value').value;
10      // Créer la requête SOAP
11      let soap_request =
12        '<?xml version="1.0" encoding="utf-8"?>' +
13        '<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance'
14        'xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-
15        envelope">' +
16        '  <soap12:Body>' +
17        '    <CelsiusToFahrenheit xmlns="https://www.w3schools.com/xml/">' +
18        '      <Celsius>' + form_celsius + '</Celsius>' +
19        '    </CelsiusToFahrenheit>' +
20        '  </soap12:Body>' +
21        '</soap12:Envelope>';
22
23      xmlhttp.onreadystatechange = function () {
24        if (xmlhttp.readyState == 4) {
25          if (xmlhttp.status == 200) {
26            console.log(xmlhttp.responseText)
27            result = document.getElementById('result')
28            result.innerHTML = xmlhttp.responseText
29          }
30        }
31      }
32      // Requête en POST avec XML en type de contenu
33      xmlhttp.setRequestHeader('Content-Type', 'text/xml; charset=utf-8');
34      xmlhttp.send(soap_request);
35    }
36  </script>
37 </head>
38 <body>
39   <form name="soap-client" action="" method="POST">
40     <label for="value">Valeur en Celsius</label>
41     <input type="text" name="value" id="value" />
42     <br />
43     <input type="button" value="Lancer la requête SOAP" onclick="soap();" />
44   </form>
45   <pre lang="xml" id="result"></pre>
46 </body>

```

<sup>1</sup> <https://www.w3schools.com/xml/tempconvert.asmx>

```
45 </html>
```

Pour essayer ce code sur votre ordinateur, il faudra installer l'extension Chrome « *CORS Unblock* » et l'activer. En effet, pour des raisons de sécurité le navigateur refuse d'effectuer des requêtes `POST` à partir d'une IP locale.

### Syntaxe À retenir

Le SOAP (Simple Object Access Protocol) n'est ni simple, ni basé sur des objets.

Pour chaque API non documentée, il faudra lire le fichier WSDL et en tirer les informations suivantes :

- Le point d'entrée de l'API,
- La liste des opérations (fonctionnalités) disponibles,
- S'il y en a, les paramètres à passer et leur type.

Avec ces informations en tête, il faudra à l'aide de Javascript envoyer une requête de type `POST` qui contiendra une enveloppe SOAP : c'est au final du XML qui respecte la convention et les normes du protocole.

### Complément

Liste d'APIs publiques SOAP<sup>(EN)</sup><sup>1</sup>

## VI. Exercice : Appliquer la notion

### Question

[solution n°3 p.21]

Sur la base des exemples précédents, utilisez l'API SOAP suivante<sup>2</sup> pour retrouver la liste des pays du Monde par leur nom.

Quel est le nom du 15<sup>ième</sup> pays dans la liste retournée par l'API ?

### Indice :

Adaptez la méthode utilisée dans le cours et dans la vidéo d'exemple pour parvenir au résultat. Au final seule l'**opération** change.

## VII. Autres façons de procéder avec fetch()

Maintenant, JavaScript intègre un moyen qui lui est propre pour créer des requêtes API. Il s'agit de l'API Fetch<sup>3</sup>. C'est une librairie permettant d'envoyer des requêtes HTTP de manière asynchrone, un peu comme les requêtes XMLHttpRequest. L'API Fetch fonctionne avec les promesses<sup>4</sup> JavaScript.

### Méthode Premier pas avec la syntaxe de l'API Fetch - requête GET

L'API Javascript Fetch est vraiment simple à utiliser. Il vous suffit d'utiliser la fonction `fetch()` avec une URL pour effectuer une requête GET.

```
1 <script>
2
3   const url = 'https://api.jokes.one/jod/';
4
5   fetch(url)
```

1 <https://documenter.getpostman.com/view/8854915/Szf26WHn?version=latest>

2 <http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso>

3 [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API)

4 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using\\_promises](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using_promises)

```

6  .then((response) => response.json())
7  .then(function(data) {
8      let jokes = data.contents.jokes;
9      console.log(jokes);
10  })
11  .catch(function(error) {
12      console.log(error);
13  });
14
15 </script>
16

```

L'URL que l'on passe va donc être recherchée via AJAX à l'aide de la fonction `fetch()`. Cette fonction renvoie une promesse que l'on récupère avec le bloc `then()`.

```

1  .then((response) => response.json())

```

Dans ce bloc, la variable `response` contient toutes les données de la requête HTTP, le header, le body et les paramètres. On utilise ici la fonction `response.json()` pour convertir l'objet renvoyé en JSON.

Dans le bloc `then` suivant, on récupère donc un objet `data` sous forme JSON que l'on peut ensuite utiliser pour alimenter notre page web.

Si pour une raison X la fonction `fetch()` retourne une erreur, on peut les récupérer avec le bloc `catch()`. Dans cette fonction on obtient une variable `error` qui va nous permettre de traiter et d'obtenir des informations.

### Complément

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 la clé : `name` et la valeur : `Sara`.

```

1  const url = 'https://randomuser.me/api';
2
3  let data = {
4      name: 'Sara'
5  }
6
7  let fetchData = {
8      method: 'POST',
9      body: data,
10     headers: new Headers()
11  }
12
13  fetch(url, fetchData)
14  .then(function() {
15      // Gérer la réponse du serveur
16  });

```

L'API Fetch fournit une interface pour la récupération de ressources. Elle paraîtra familière à tout utilisateur de XMLHttpRequest<sup>1</sup>, mais cette nouvelle API propose néanmoins un ensemble de fonctionnalités plus souples et plus puissantes. Il s'agit d'une superbe alternative à XMLHttpRequest mais demande au préalable une bonne étude de la documentation de l'API.

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



## VIII. Essentiel

## IX. Auto-évaluation

### A. Exercice

#### Question

[solution n°4 p.22]

Vous allez devoir développer un client REST complet capable de gérer les actions en GET, POST et DELETE de l'API ouverte « *Postman Echo* » disponible à cette adresse<sup>1</sup>.

Votre client Javascript doit être capable de récupérer de l'information en GET, et d'envoyer de l'information en POST et en DELETE.

Vous devez afficher pour chacun des appels la réponse HTTP retournée par l'API de Postman.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.24]

##### Question 1

Les API sont classées en 2 familles principales.

- ☐ Les API ouvertes
- ☐ Les API fermées
- ☐ Les API SOAP

##### Question 2

Quelle méthode utiliseriez-vous pour récupérer une liste de produits ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☐ DELETE

##### Question 3

Quelle méthode utiliseriez-vous pour supprimer une ressource ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☐ DELETE

##### Question 4

---

<sup>1</sup> <https://docs.postman-echo.com/>

Quelle méthode utiliseriez-vous pour créer un nouvel utilisateur au travers d'une API REST ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☐ DELETE

Question 5

Combien de niveaux contient l'échelle de Richardson ?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Question 6

Quel est le format utilisé pour communiquer de l'information quand on utilise une API SOAP ?

- ☐ Le JSON
- ☐ Le HTML
- ☐ Le XML

Question 7

Quel est le format du fichier de définition d'une API SOAP ?

- ☐ XML
- ☐ WSDL
- ☐ Les 2 !

Question 8

Comment trouver **à coup sûr** le point d'entrée d'une API SOAP ?

- ☐ En lisant la documentation
- ☐ En consultant le fichier WSDL
- ☐ Les 2 !

Question 9

Comment connaître **à coup sûr** le nombre d'opérations disponibles pour un serveur SOAP ?

- ☐ En lisant la documentation
- ☐ En consultant le fichier WSDL
- ☐ Les 2 !

Question 10

Pour une opération/fonctionnalité donnée d'un serveur SOAP, quel nœud du fichier WSDL consulter pour avoir une idée de ce que le serveur attend comme paramètres, pour vous apporter la réponse demandée ?

- ☐ `wSDL:definitions`
- ☐ `wSDL:parameters`
- ☐ `wSDL:types;`

## Solutions des exercices



**p. 5 Solution n°1**

Le personnage avec l'identifiant n°12 est « *Balon Greyjoy* », le seigneur des Îles de Fer !

Voici un code fonctionnel (à intégrer dans une page HTML) :

```
1 const request = new XMLHttpRequest();
2 request.open('GET', 'https://cat-fact.herokuapp.com/facts', true);
3
4 request.onload = function() {
5   if (this.status >= 200 && this.status < 400) {
6     // Accès à l'API réussi !
7     let personnages = JSON.parse(this.response);
8     console.log(personnages[11]['name']);
9   } else {
10    // Erreur retournée par le serveur
11  }
12 }
13 };
14
15 request.onerror = function() {
16   // Problème de connexion
17 };
18
19 request.send();
```

**p. 9 Solution n°2**

Le premier lancement de l'entreprise Space X a été effectué le 24 mars 2006.

Voici le code qui vous aurait permis de trouver ce résultat :

```
1 const request = new XMLHttpRequest();
2 request.open('GET', 'https://api.spacexdata.com/v3/launches/past');
3
4 request.onload = function() {
5   response_from_server = JSON.parse(request.responseText)[0];
6   mission_name = document.getElementById('mission_name');
7   date = document.getElementById('date');
8
9   console.log(response_from_server.mission_name)
10  // Charger le contenu
11  console.log(new Date(response_from_server.launch_date_utc).toLocaleDateString("fr-FR"));
12 }
13 request.send();
```

Et pour l'histoire, cette mission a malheureusement échoué !

**p. 15 Solution n°3**

Et la réponse est l'Autriche (Austria).

Voici un code fonctionnel permettant de retrouver ce résultat :

```

1 <html>
2 <head>
3   <title>Liste de pays en utilisant SOAP en JavaScript</title>
4   <script type="text/javascript">
5     function soap() {
6       const xmlhttp = new XMLHttpRequest();
7       xmlhttp.open('POST',
8         'http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso', true);
9
10      // Créer la requête SOAP
11      let soap_request =
12        '<?xml version="1.0" encoding="utf-8"?>' +
13        '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">' +
14        '  <soap:Body>' +
15        '    <ListOfCountryNamesByName
16      xmlns="http://www.oorsprong.org/websamples.countryinfo">' +
17        '  </ListOfCountryNamesByName>' +
18        '  </soap:Body>' +
19        '</soap:Envelope>';
20
21      xmlhttp.onreadystatechange = function () {
22        if (xmlhttp.readyState == 4) {
23          if (xmlhttp.status == 200) {
24            console.log(xmlhttp.responseText)
25            result = document.getElementById('result')
26            result.innerHTML = xmlhttp.responseText
27          }
28        }
29      }
30      // Requête en POST avec XML en type de contenu
31      xmlhttp.setRequestHeader('Content-Type', 'text/xml; charset=utf-8');
32      xmlhttp.send(soap_request);
33    }
34  </script>
35 </head>
36 <body>
37   <form name="soap-client" action="" method="POST">
38     <input type="button" value="Lancer la requête SOAP" onclick="soap();" />
39   </form>
40   <pre lang="xml" id="result"></pre>
41 </body>
42 </html>

```

#### p. 17 Solution n°4

Il faudra, pour exécuter ce projet en local, installer une extension navigateur appelée « Cross Domain<sup>1</sup> » car l'API Postman Echo interdit les appels à son service à partir d'une IP locale.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Final project</title>

```

1 <https://chrome.google.com/webstore/detail/cross-domain-cors/mjhpgnbimicffchbodmgfnemoghjakai>

```

6     </head>
7     <body>
8         <h1>Client REST pour Postman Echo</h1>
9
10        <div class="rest_block">
11            <form method="POST" action="">
12                <label for="get_data">Données en GET</label>
13                <input id="get_data" name="get_data" />
14                <input type="button" id="get_endpoint" value="Envoyer une requête en GET" />
15                <br />
16                <h2>Résultats Appel en GET</h2>
17                <textarea id="result_get"></textarea>
18            </form>
19        </div>
20
21        <div class="rest_block">
22            <form method="POST" action="">
23                <label for="post_data">Données en POST</label>
24                <input id="post_data" name="post_data" />
25                <input type="button" id="post_endpoint" value="Envoyer une requête en POST" />
26                <br />
27                <h2>Résultats Appel en POST</h2>
28                <textarea id="result_post"></textarea>
29            </form>
30        </div>
31
32        <div class="rest_block">
33            <form method="POST" action="">
34                <label for="delete_data">Données en DELETE</label>
35                <input id="delete_data" name="delete_data" />
36                <input type="button" id="delete_endpoint" value="Envoyer une requête en DELETE"
37            />
38                <br />
39                <h2>Résultats Appel en DELETE</h2>
40                <textarea id="result_delete"></textarea>
41            </form>
42        </div>
43        <script>
44            const endpoint = 'https://postman-echo.com/'
45
46            // Les fonctions d'appel
47            function api_get(data) {
48                const request = new XMLHttpRequest();
49                request.open('GET', `${endpoint}get?${data}`, false);
50                request.setRequestHeader('Content-type', 'application/json');
51                request.send();
52                response_from_api = JSON.parse(request.responseText);
53
54                return JSON.stringify(response_from_api.args);
55            }
56
57            function api_post(data) {
58                const request = new XMLHttpRequest();
59                request.open('POST', `${endpoint}post`, false);
60                request.setRequestHeader('Content-type', 'application/json');
61                request.send(data);
62
63                response_from_api = JSON.parse(request.responseText, true);

```

```

63         return JSON.stringify(response_from_api.args);
64     }
65
66     function api_delete() {
67         const request = new XMLHttpRequest();
68         request.open('DELETE', `${endpoint}patch`, false);
69         request.setRequestHeader('Content-type', 'application/json');
70         request.send();
71         response_from_api = JSON.parse(request.responseText, true);
72         return JSON.stringify(response_from_api.args);
73     }
74
75     // Gestion de la soumission des formulaires
76     const get_form = document.getElementById('get_endpoint');
77     const post_form = document.getElementById('post_endpoint');
78     const delete_form = document.getElementById('delete_endpoint');
79
80     // Appels en GET
81     get_form.addEventListener('click', () => {
82         const input_data = document.getElementById('get_data').value;
83         const textarea = document.getElementById('result_get');
84
85         textarea.innerHTML = api_get(input_data);
86     });
87
88     // Appels en POST
89     post_form.addEventListener('click', () => {
90         const input_data = document.getElementById('post_data').value;
91         const textarea = document.getElementById('result_post');
92
93         textarea.innerHTML = api_get(input_data);
94     });
95
96     // Appels en DELETE
97     delete_form.addEventListener('click', () => {
98         const input_data = document.getElementById('delete_data').value;
99         const textarea = document.getElementById('result_delete');
100
101         textarea.innerHTML = api_get(input_data);
102     });
103
104     </script>
105 </body>
106 </html>

```


## Exercice p. 17 Solution n°5

### Question 1

Les API sont classées en 2 familles principales.

- ☒ Les API ouvertes
- ☒ Les API fermées
- ☐ Les API SOAP



-  Toutes les API existantes sont ouvertes ou fermées. Les API fermées nécessitent de s'authentifier auprès du serveur pour être autorisées à interagir avec les ressources.

### Question 2

---

Quelle méthode utiliseriez-vous pour récupérer une liste de produits ?

- ☒ GET
- ☐ POST
- ☐ PUT
- ☐ DELETE


-  La méthode GET est recommandée pour récupérer une liste de ressources au travers d'une API REST.

### Question 3

---

Quelle méthode utiliseriez-vous pour supprimer une ressource ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☒ DELETE


-  La méthode DELETE est recommandée pour supprimer une ressource au travers d'une API REST.

### Question 4

---

Quelle méthode utiliseriez-vous pour créer un nouvel utilisateur au travers d'une API REST ?

- ☐ GET
- ☒ POST
- ☐ PUT
- ☐ DELETE


-  La méthode POST est recommandée pour créer une ressource au travers d'une API REST.

### Question 5

---


Combien de niveaux contient l'échelle de Richardson ?

- ☐ 1
- ☐ 2
- ☐ 3
- ☒ 4

-  L'échelle de Richardson part du niveau 0 au niveau 3 : il y a donc bien 4 niveaux !


### Question 6

Quel est le format utilisé pour communiquer de l'information quand on utilise une API SOAP ?

- ☐ Le JSON
- ☐ Le HTML
- ☒ Le XML
-  Contrairement aux API REST qui n'imposent pas de format particulier, les API SOAP ne fonctionnent qu'avec le langage XML.


### Question 7

Quel est le format du fichier de définition d'une API SOAP ?

- ☐ XML
- ☐ WSDL
- ☒ Les 2 !
-  La définition d'une API SOAP se fait dans un fichier WSDL, qui est un fichier XML qui respecte des conventions spécifiques à la définition des services web SOAP.


### Question 8

Comment trouver **à coup sûr** le point d'entrée d'une API SOAP ?

- ☐ En lisant la documentation
- ☒ En consultant le fichier WSDL
- ☐ Les 2 !
-  Le fichier WSDL contient plusieurs nœuds XML `wsdl:address` qui listent le nombre de points d'entrée disponibles pour les fonctionnalités partagées par le service web.

### Question 9

Comment connaître **à coup sûr** le nombre d'opérations disponibles pour un serveur SOAP ?

- ☐ En lisant la documentation
- ☒ En consultant le fichier WSDL
- ☐ Les 2 !
-  Le fichier WSDL contient plusieurs nœuds XML `wsdl:operation` qui listent chaque opération disponible pour les fonctionnalités partagées par le service web.

### Question 10

Pour une opération/fonctionnalité donnée d'un serveur SOAP, quel nœud du fichier WSDL consulter pour avoir une idée de ce que le serveur attend comme paramètres, pour vous apporter la réponse demandée ?

- ☐ `wsdl:definitions`
- ☐ `wsdl:parameters`
- ☒ `wsdl:types;`

Q Si une opération nécessite des paramètres, ils seront définis dans le fichier WSDL. Une documentation technique peut être obsolète, mais ce fichier est toujours juste !