|  |
| --- |
| **CER U.E. 05 [API]** |
| **[BENAMEUR Fares]** |

**Equipe d’animation :**

**Animateur :ASSAM Meriem**

**Scribe : SICHAIB Amina**

**Secrétaire : OUANOUGHI Noufel**

**Gestionnaire : ELBASGHI Meriem**

**Analyse du contexte :**

Une équipe envisage d'utiliser une API externe pour remplir automatiquement certains champs d'un formulaire client, mais des problèmes surviennent

**​**

**​Définition de la problématique :**

Comment simplifier le formulaire tout en assurant la sécurité, y compris l'intégration d'une API pour l'autocomplétion des champs ?

1. **Mots à Définir**

* **[Fichier JSON] :**
* Un fichier JSON (JavaScript Object Notation) est un format de données léger et facile à lire et à écrire, qui est largement utilisé pour l'échange de données sur le web et dans de nombreux systèmes informatiques. Il est basé sur une syntaxe JavaScript simplifiée et est souvent utilisé pour représenter des structures de données simples.
* **[XML] :**
* **XML (eXtensible Markup Language) est un langage de balisage conçu pour stocker et transporter des données de manière lisible par les humains et les machines. Il est similaire à HTML (HyperText Markup Language), mais contrairement à HTML qui est principalement utilisé pour la création de pages web et la présentation de contenu, XML est utilisé pour structurer et stocker des données de manière plus générique.**
* **[API Publique] : Une API publique (Application Programming Interface) est une interface de programmation mise à disposition par une organisation ou un service pour permettre aux développeurs tiers d'accéder à certaines fonctionnalités ou données.**
* **[Endpoint] :**
* **Un endpoint (point de terminaison) est une URL spécifique à une API (Application Programming Interface) qui est utilisée pour accéder à une fonctionnalité ou à des données spécifiques fournies par cette API. Chaque endpoint représente une ressource ou une opération particulière exposée par l'API.**

1. **Mots Clés**

* **[Formulaire]:**
* **[URI] :**
* **[HTTP] :**

1. **Plan D’action**
   1. **[Etudier les API] :**
2. **Qualification :** abouti
3. **Démonstration :**

## API : définition

Une API, ou interface de programmation d'application, est un ensemble de définitions et de protocoles qui facilite la création et l'intégration des applications.

[**Le manuel d'utilisation des API : 7 meilleures pratiques des équipes qui réussissent à exploiter les API**](https://www.redhat.com/fr/resources/api-owners-manual-ebook)

### Types d'API

#### API privées

L'API n'est utilisable qu'en interne. Cette approche permet de garder un contrôle total sur l'API.

#### API partenaires

L'API est partagée avec certains partenaires de l'entreprise. Cette approche peut générer de nouveaux flux de revenus sans compromettre la sécurité.

#### API publiques

L'API est accessible à tous. Cette approche autorise les tiers à développer des applications qui interagissent avec votre API et peut devenir source d'innovations.

## Innovation par les API

En rendant vos API accessibles à vos partenaires ainsi qu'aux tiers, vous pouvez :

* générer de nouveaux canaux de revenus ou étendre ceux qui existent déjà ;
* étendre la portée de votre marque ;
* stimuler l'innovation [Open Source](https://www.redhat.com/fr/about/open-source) ou améliorer l'efficacité grâce au développement et à la collaboration externes.

Intéressant, non ? Mais quel est le rôle des API dans tout ça ?

### Cas d'utilisation des API

Reprenons notre exemple du distributeur de livres.

Imaginez qu'un partenaire de l'entreprise développe une application qui permet aux clients de localiser les livres qu'ils cherchent dans les rayons. En améliorant l'expérience client, cette application va attirer d'autres clients dans la librairie, elle-même cliente du distributeur, qui en fin de compte aura étendu son canal de revenus.

Une entreprise tierce peut aussi utiliser une API publique pour développer une application afin que les clients puissent acheter des livres directement auprès du distributeur, sans passer par la librairie. Dans ce cas, le distributeur aura ouvert un nouveau canal de revenus.

Une entreprise peut tirer parti du partage de ses API avec certains de ses partenaires ou avec le monde entier. Chaque partenariat vous permet d'étendre la notoriété de votre marque en parallèle de vos campagnes marketing. En rendant vos technologies publiques, avec une API publique par exemple, vous encouragez les développeurs à créer un écosystème d'applications autour de votre API. Plus les gens utilisent vos technologies, plus ils sont susceptibles de faire affaire avec vous.

Vous pouvez ainsi obtenir des résultats aussi inattendus qu'intéressants en ouvrant l'accès à vos technologies et ces résultats peuvent parfois bouleverser un secteur tout entier. Dans le cas de notre distributeur de livres, de nouvelles entreprises, comme un service de location de livres, peuvent provoquer un changement fondamental dans son fonctionnement. Les API publiques et partenaires vous permettent de profiter des innovations d'une communauté de développeurs plus large. Les idées novatrices peuvent germer n'importe où. Les entreprises doivent se tenir au courant des changements qui s'opèrent sur leur marché et se préparer à y faire face. C'est là que les API interviennent.

[**Le gouvernement irlandais améliore la productivité grâce aux API**](https://www.redhat.com/fr/topics/api/node/705611)

## L'histoire des API en quelques mots

Les API sont apparues à l'aube de l'informatique, avant même les ordinateurs personnels. À cette époque, elles étaient surtout utilisées en tant que bibliothèques pour les systèmes d'exploitation. Elles résidaient presque toutes en local sur les systèmes sur lesquels elles s'exécutaient, même si elles transféraient parfois des messages entre les mainframes. Presque 30 ans après, les API sont sorties de leurs environnements locaux. Au début des années 2000, elles sont devenues importantes pour l'intégration des données à distance.

## API distantes

Les API distantes sont conçues pour interagir via un réseau de communication. Ici, « distant » signifie que les ressources manipulées par l'API ne se trouvent pas sur l'ordinateur qui formule la requête. Le réseau de communication le plus fréquemment utilisé étant Internet, la plupart des API sont conçues sur la base des normes web. Toutes les API distantes ne sont pas des API web, mais on peut supposer que toutes les API web sont distantes.

Les API web utilisent en général le protocole HTTP pour leurs messages de requête et fournissent une définition de la structure des messages de réponse. Les messages de réponse se présentent la plupart du temps sous la forme d'un fichier XML ou JSON. Ces deux formats sont les plus courants, car les données qu'ils contiennent sont faciles à manipuler pour les autres applications.

## API SOAP ou REST

Pour standardiser l'échange des informations entre les API toujours plus nombreuses, il a fallu développer un protocole : le « Simple Object Access Protocol », plus connu sous le nom de SOAP. Les API conçues d'après le protocole SOAP utilisent le format XML pour leurs messages et reçoivent des requêtes via HTTP ou SMTP. SOAP a pour objectif de simplifier l'échange des informations entre les applications qui s'exécutent dans des environnements différents ou qui ont été écrites dans des langages différents.

Le [« Representational State Transfer », ou REST,](https://www.redhat.com/fr/topics/api/what-is-a-rest-api)est une autre tentative de normalisation. Les API web qui respectent les contraintes de l'architecture REST sont appelées API RESTful. Ces deux éléments diffèrent sur un point fondamental : SOAP est un protocole, alors que REST est un style d'architecture. Cela signifie qu'il n'existe aucune norme officielle qui régit les API web RESTful. Selon la définition proposée par Roy Fielding dans sa thèse « Architectural Styles and the Design of Network-based Software Architectures », les API sont RESTful tant qu'elles respectent les six contraintes de conception d'un système RESTful :

* **Architecture client-serveur :** une architecture REST est composée de clients, de serveurs et de ressources et elle traite les requêtes via le protocole HTTP.
* **Serveur stateless :** le contenu du client n'est jamais stocké sur le serveur entre les requêtes. Les informations sur l'état de la session sont, quant à elles, stockées sur le client.
* **Mémoire cache :** la mise en mémoire cache permet de se passer de certaines interactions entre le client et le serveur.
* **Système à couches :** des couches supplémentaires peuvent assurer la médiation dans les interactions entre le client et le serveur. Ces couches peuvent remplir des fonctions supplémentaires, telles que l'équilibrage de charge, le partage des caches ou la sécurité.
* **Code à la demande (facultatif) :** un serveur peut étendre les fonctionnalités d'un client en lui transférant du code exécutable.
* **Interface uniforme :** cette contrainte est capitale pour la conception des API RESTful et couvre quatre aspects différents :
  + **Identification des ressources dans les requêtes :** les ressources sont identifiées dans les requêtes et sont séparées des représentations retournées au client.
  + **Manipulation des ressources par des représentations :** les clients reçoivent des fichiers qui représentent les ressources. Ces représentations doivent contenir suffisamment d'informations pour être modifiées ou supprimées.
  + **Messages autodescriptifs :** tous les messages renvoyés au client contiennent assez d'informations pour décrire la manière dont celui-ci doit traiter les informations.
  + **Hypermédia comme moteur du changement des états applicatifs :** après avoir accédé à une ressource, le client REST doit être en mesure de découvrir toutes les autres actions disponibles par des hyperliens.

Ces contraintes peuvent sembler difficiles à appliquer, mais dans les faits, elles le sont moins qu'un protocole. C'est pour cette raison que les API RESTful prennent progressivement le pas sur les API SOAP.

Ces dernières années, la spécification OpenAPI s'est imposée comme la norme commune pour définir les API REST. La norme OpenAPI permet aux développeurs de créer des interfaces d'API REST indépendantes du langage de manière à ce que les utilisateurs puissent les comprendre avec un minimum d'approximation.

Une autre norme d'API est en train d'émerger : [GraphQL](https://www.redhat.com/fr/topics/api/what-is-graphql), un langage de requête et un environnement d'exécution côté serveur qui se propose de remplacer l'architecture REST. GraphQL s'attache à fournir aux clients uniquement les données qu'ils ont demandées, et rien de plus. Utilisé à la place de REST, GraphQL permet aux développeurs de créer des requêtes qui extraient les données de plusieurs sources à l'aide d'un seul appel d'API.

[**En savoir plus sur les différences entre SOAP et REST**](https://www.redhat.com/fr/topics/integration/whats-the-difference-between-soap-rest)

## Architectures SOA et de microservices

Les deux approches architecturales qui utilisent le plus les API distantes sont l'[architecture orientée services (SOA)](https://www.redhat.com/fr/topics/cloud-native-apps/what-is-service-oriented-architecture) et l'architecture de microservices. La SOA est l'approche la plus ancienne. Elle visait à l'origine à améliorer les applications monolithiques. Par définition, une application monolithique fait tout. Mais certaines fonctions peuvent être fournies par d'autres applications faiblement couplées via un modèle d'intégration, comme un ESB (Enterprise Service Bus).

Si l'architecture SOA est plus simple qu'une architecture monolithique sous bien des aspects, elle peut potentiellement provoquer des changements en cascade au sein de l'environnement si les interactions entre les différents composants ne sont pas parfaitement comprises. En complexifiant l'environnement, l'architecture SOA réintroduit une partie des problèmes qu'elle cherchait à résoudre.

Les architectures de microservices fonctionnent d'une manière très similaire, dans le sens où elles utilisent des services faiblement couplés. Par contre, elles poussent la déstructuration de l'architecture classique encore plus loin. Les services qui composent l'architecture de microservices utilisent une structure de messagerie commune, telle que les API RESTful. Ils se servent des API RESTful pour communiquer entre eux simplement, sans convertir leurs données ni recourir à des couches d'intégration supplémentaires. L'utilisation des API RESTful permet et favorise même l'accélération de la distribution des nouvelles fonctions et mises à jour. Chaque service est distinct. Vous pouvez remplacer, améliorer ou supprimer chacun d'entre eux sans affecter les autres services de l'architecture. Cette architecture légère vous aide à optimiser les ressources distribuées ou cloud et à faire évoluer chaque service de façon dynamique.

* 1. **[Etudier AJAX]**

1. **Qualification : abouti**
2. **Démonstration :**

AJAX is a developer's dream, because you can:

* Read data from a web server - after the page has loaded
* Update a web page without reloading the page
* Send data to a web server - in the background

### **AJAX Example**

# **AJAX**

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_first)

## **AJAX Example Explained**

### **HTML Page**

<!DOCTYPE html>  
<html>  
<body>  
  
<div id="demo">  
  <h2>Let AJAX change this text</h2>  
  <button type="button" onclick="loadDoc()">Change Content</button>  
</div>  
  
</body>  
</html>

The HTML page contains a <div> section and a <button>.

The <div> section is used to display information from a server.

The <button> calls a function (if it is clicked).

The function requests data from a web server and displays it:

### **Function loadDoc()**

function loadDoc() {  
  const xhttp = new XMLHttpRequest();  
  xhttp.onload = function() {  
    document.getElementById("demo").innerHTML = this.responseText;  
    }  
  xhttp.open("GET", "ajax\_info.txt", true);  
  xhttp.send();  
}

ADVERTISEMENT

## **What is AJAX?**

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

## **How AJAX Works**



* 1. An event occurs in a web page (the page is loaded, a button is clicked)
* 2. An XMLHttpRequest object is created by JavaScript
* 3. The XMLHttpRequest object sends a request to a web server
* 4. The server processes the request
* 5. The server sends a response back to the web page
* 6. The response is read by JavaScript
* 7. Proper action (like page update) is performed by JavaScript

## **Modern Browsers (Fetch API)**

Modern Browsers can use Fetch API instead of the XMLHttpRequest Object.

The Fetch API interface allows web browser to make HTTP requests to web servers.

If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way.

* 1. **[Etudier la securité des API]**

1. **Qualification : abouti**
2. **Démonstration :**

## Qu'est-ce que la sécurité des API ?

Une interface de programmation d'application ([API](https://www.cloudflare.com/learning/security/api/what-is-an-api/)) est un moyen pour un logiciel d'interagir avec un autre logiciel. Si un programme ou une application possède une API, des clients externes peuvent lui demander des services.

La sécurité des API est le processus de protection des API contre les attaques. Tout comme les applications, les réseaux et les serveurs peuvent faire l'objet d'attaques, les API peuvent être victimes d'un certain nombre de menaces différentes.

La sécurité des API est un élément essentiel de la sécurité des applications web [. La plupart des applications web modernes reposent sur des API pour fonctionner, et les API introduisent un risque supplémentaire pour une application en permettant à des parties extérieures d'y accéder. On peut comparer cette situation à celle d'une entreprise qui ouvre ses bureaux au public : la présence d'un plus grand nombre de personnes dans les locaux, dont certaines peuvent être inconnues des employés de l'entreprise, augmente le risque. De même, une API permet à des personnes extérieures d'utiliser un programme, ce qui introduit un risque supplémentaire pour l'infrastructure du service API.](https://www.cloudflare.com/learning/security/what-is-web-application-security/)

## [Quels sont les risques courants en matière de sécurité des API ?](https://www.cloudflare.com/learning/security/what-is-web-application-security/)

* [**Exploitation des vulnérabilités :** On parle d'exploitation des vulnérabilités lorsqu'un attaquant envoie des données spécialement conçues à une cible, données qui tirent parti d'une faille dans la construction de la cible. Ces failles, connues sous le nom de "vulnérabilités," peuvent donner à l'attaquant diverses formes d'accès involontaire à une API ou à son application correspondante. L'Open Web Application Security Project (](https://www.cloudflare.com/learning/security/what-is-web-application-security/)[OWASP](https://www.cloudflare.com/learning/security/threats/owasp-top-10/)) tient à jour une liste des [10 principales vulnérabilités des API](https://www.cloudflare.com/learning/security/api/owasp-api-security-top-10/), comme [l'injection SQL](https://www.cloudflare.com/learning/security/threats/sql-injection/), la mauvaise configuration de la sécurité, etc. Si un exploit cible une vulnérabilité inconnue jusqu'alors, il s'agit d'une [menace de type "zero-day"](https://www.cloudflare.com/learning/security/threats/zero-day-exploit/) - de telles menaces sont extrêmement difficiles à arrêter.
* **Attaques basées sur l'authentification :** Les clients doivent s'authentifier avant de pouvoir effectuer des demandes d'API afin que le serveur d'API n'accepte pas de demandes provenant de sources inconnues ou illégitimes. Il existe plusieurs façons de procéder, mais chacune d'entre elles est susceptible d'être compromise. Par exemple, un attaquant peut obtenir les informations d'identification d'un client légitime, voler une clé d'API ou intercepter et utiliser un jeton d'authentification.
* **Erreurs d'autorisation :** L'autorisation détermine le niveau d'accès de chaque utilisateur. Si les autorisations ne sont pas gérées avec soin, un client API peut avoir accès à des données auxquelles il ne devrait pas avoir accès, ce qui augmente le risque d'une violation de données .
* **Attaques DoS et DDoS :** Un trop grand nombre de demandes adressées à une API peut ralentir ou arrêter le service pour les autres clients. Certains attaquants dirigent volontairement un grand nombre de requêtes vers une API dans le cadre d'une attaque par déni de service (DoS) ou par déni de service distribué (DDoS) .

Les stratégies de sécurité des API peuvent contribuer à atténuer ces risques et d'autres encore.

Des mesures d'authentification et d'autorisation fortes permettent de s'assurer que les données ne fuient pas et que seuls les clients autorisés effectuent des demandes d'API. La protection contre les attaques DDoS et la limitation du débit peuvent mettre fin aux attaques DDoS. La validation des schémas et l'utilisation d'un pare-feu d'application web (WAF) peuvent bloquer les exploits de vulnérabilité.

* 1. **[Simplifier la saisie du formulaire]**

1. **Qualification : abouti**

**Démonstration : fichiers en piece jointe**