

UE AD FIL A1

UE Architectures et microservices FISE A3 LOGIN

API REST

2026-2027

Hélène Coullon



IMT Atlantique

Bretagne-Pays de la Loire

École Mines-Télécom

Table of Contents

1. Avant de démarrer
2. API REST
3. Documenter son API REST

Avant de démarrer

HyperText Transfer Protocol

Développé par Tim Berners-Lee au CERN en 1989 (Suisse) avec d'autres concepts qui ont servi de base à la création du World Wide Web (HTML et URI)

- Protocole de communication réseau
- Couche applicative du modèle OSI
- Protocole de type client/serveur requête/réponse

Une requête HTTP est constituée des éléments suivants:

- la ligne de requête : `METHOD URL PROTOCOL-VERSION`
- des éléments d'en-tête optionnels
- un corps de message optionnel

Les méthodes les plus utilisées sont les suivantes :

- `GET` : obtenir une ressource située à l'URL spécifiée
- `POST` : envoi de données à l'URL spécifiée
- `PUT` : envoi de données à l'URL spécifiée (typiquement pour une mise à jour)
- `DELETE` : suppression de la ressource à l'URL spécifiée

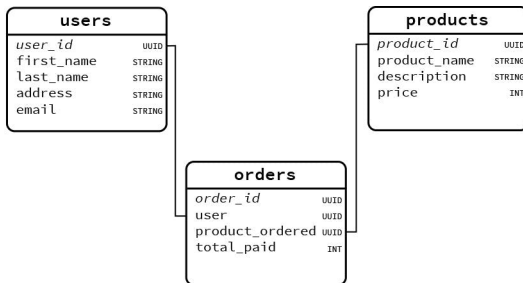
Les versions de HTTP (1/2)

- 1991 HTTP/0.9
 - permet de récupérer le contenu (GET) d'une page HTML uniquement
 - la connexion TCP/IP est toujours fermée après l'envoi de la réponse
- 1996 HTTP/1.0
 - permet de récupérer (GET) tout type de fichiers
 - la connexion TCP/IP est toujours fermée après l'envoi de la réponse
- 1997 HTTP/1.1
 - conservation de la connexion
 - pipelining (plusieurs requêtes avant la réponse)
 - permet d'envoyer des informations au serveur (POST/PUT)
 - TRACE (suivi du chemin entre le client et le serveur)
 - caching (conservation du contenu en mémoire tampon)
 - ...

- 2015 HTTP/2.0
 - données binaires au lieu de texte
 - requêtes parallèles côté client et serveur
 - compression des en-têtes de messages
 - PUSH (envoi en cash client de données à l'avance)
- 2020 HTTP/3.0 (draft)
 - UDP au lieu de TCP

Sérialiser des données ?

Comment envoyer des données complexes sur le réseau ?



Il faut envoyer des données linéaires, une suite d'informations.

- Formats **standardisés** de données **textuelles**
- Représentation structurée (arbres) de l'information textuelle
- Facilement lisible par un être humain
- Formats **sérialisables**
 - communication sur le réseau
- Formats comparables à XML mais plus légers

JSON

- JavaScript Object Notation
- Exemples d'utilisation :
 - Requêtes et réponses HTTP
 - Bases de données NoSQL

YAML

- Yet Another Markup Language - (version 1.0)
- YAML Ain't Markup Language - (version 1.1)
- Exemples d'utilisation :
 - Fichiers configuration
 - Spécifications

- Valeurs atomiques
 - Des chaînes de caractères
 - Des nombres
 - 3 constantes : true, false et null
- Paires clé/valeur
 - La clé est un string
 - La valeur est une valeur atomique
 - Le nom et la valeur d'une paire sont séparés par un deux-points
- 2 types de valeurs composites
 - Collection de paires clé/valeur = objet, dictionnaire, enregistrement
 - Une séquence de valeurs = un tableau, un vecteur, une liste

[JSON exemple](#) and [YAML exemple](#)

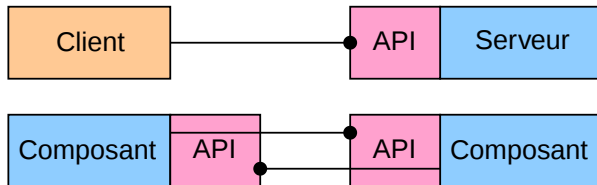
Transformation de JSON à YAML et vice-versa

API REST

- Coder à la main l'utilisation de protocoles réseau ?
 - Implémenter un client et serveur TCP/IP ?
 - Besoin de monter en **abstractions** !
- Comment connaître les interactions possibles avec les autres composants ?
 - Besoin d'une interface ! (**API**)

Utilisation de **bibliothèques et de frameworks** pour régler ces questions !

Application Programming Interface



Accord ou **contrat** entre un client et un serveur sur les interactions possibles avec le serveur

REpresentational State Transfer (Roy Fielding 2000)

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Tirer profit de la flexibilité du web pour le design d'applications et de systèmes distribués.

- Penser son application/système comme une page web.
- Les ressources sont demandées au serveur sous la forme d'une requête HTTP
- Une ressource correspond à une URL
- REST utilise les méthodes standardisées de HTTP/1.1

Ex : GET <http://adress/resource> HTTP/1.1

Tutoriel Flask

Documenter son API REST

Pourquoi documenter son API ?

- Source de référence précise sur les interactions possibles avec le serveur
 - cahier des charges, spécification, contrat
- Outil et guide pour les utilisateurs de l'API
- Meilleure adoption de l'API par les utilisateurs

Pourquoi suivre un standard ?

- Niveau d'information adequat et standardisés
- Compréhensible des humains connaissant le standard
- Plus facilement compréhensible pour des outils automatiques

Exemples de standards

- OpenAPI (anciennement Swagger)
 - <https://swagger.io/docs/specification/about/>
- RAML
 - <https://raml.org/>

Tutoriel OpenAPI

A vous de jouer !

- Installation des environnements
- Rejouer les tutos dans votre coin
- Passer au TP REST