SIO	API REST - Définition

Dans un monde de plus en plus régi par la *data* et l'interactivité entre applications, les APIs ont pris une place prépondérante dans le développement informatique. Concevoir, développer ou maintenir une **API REST** sont aujourd'hui parmi les missions les plus communes pour un développeur back-end ou fullstack. C'est pourquoi la compréhension du concept d'API et de ses différents standards est primordiale pour un développeur, tout comme sa capacité à en construire.
Pour une courte introduction, voir la vidéo de Cookie connecté.

Vocabulaire

API : Application Programming Interface / Interface de programmation d'application
C'est un programme qui permet à des sites extérieurs de pouvoir accéder aux données/services d'un site distant.

REST Representational State Transfer / Transfert d'état représentatif
C'est une norme de transfert de données entre 2 applications.

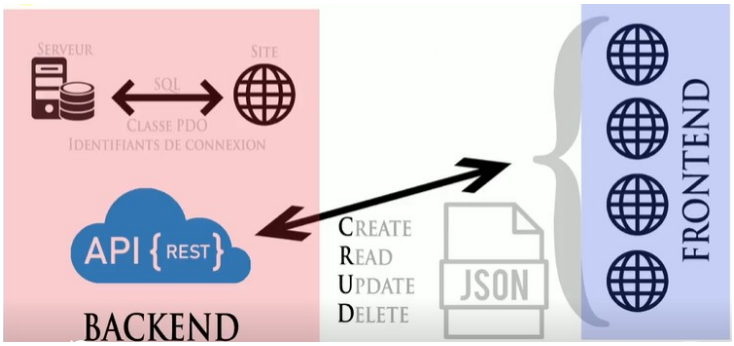
API RESTFUL

C'est une architecture et non un protocole, qui est née d'un besoin de simplifier l'accès aux données des différents services, tout en assurant une parfaite sécurité. En effet cette solution permet de donner des accès sur les opérations CRUD à une base de données sans jamais donner d'identifiants de connexion à la base de données. Ceci évite le :

- risque de piratage,
- risque de modifications,
- obligation de créer des comptes génériques pour gérer les droits,
- …

La solution est un ensemble de codes réagissant aux requêtes définies dans le protocole HTTP.

L'intérêt est de permettre à n'importe quel client, d'accéder aux données que l'on veut rendre accessibles. La description de ces données utilise le format JSON, pour ne plus être dépendant de l'encodage de chaque base de données.



Source : C. Brison

On a donc 2 parties distinctes :

BackEnd : qui est la création de l'API Rest qu'on va mettre à la disposition des utilisateurs.

FrontEnd : qui va utiliser une API pour interagir avec les données mises à notre disposition.

Règles et syntaxe

Pour créer et/ou utiliser un service RESTful, il faut respecter un certain nombre de règles. Il en existe d'autres que celles citées ci-dessous, mais ces dernières seront suffisantes pour nous.

1-URI = identifiant pour les ressources

Une URI sert de point d'entrée pour un type de ressource. Il est important de formater les URI de manière logique, en utilisant correctement la syntaxe des URLs HTTP.

2-Méthodes HTTP pour les opérations

Le protocole HTTP propose plusieurs méthodes à utiliser qu'il faut respecter :

- **POST** qui permet la création (Create <=> INSERT INTO)
- **GET** qui permet la lecture (Read <=> SELECT)
- **PUT** qui permet la mise à jour (Update <=> UDATE)
- **DELETE** qui permet la suppression (Delete <=> DELETE FROM)

Il faut donc en tant que frontend, utiliser ces méthodes pour interroger le serveur backend et permettre de faire les opérations CRUD sur la base de données.

Imaginons une application permettant d'agir sur la table "personne" dans une base de données, on pourrait trouver les liens suivants dans l'API :

- GET /api/personnes renvoyer la liste des personnes
- GET /api/personne/5 renvoyer la personne N°5
- PUT /api/personne mettre à jour la personne (données dans le message format Json)
- DELETE /api/personne effacer la personne (données dans le message format Json)
- POST /api/personne ajouter une personne (données dans message format Json)

Chacun lien s'appelle endpoint.

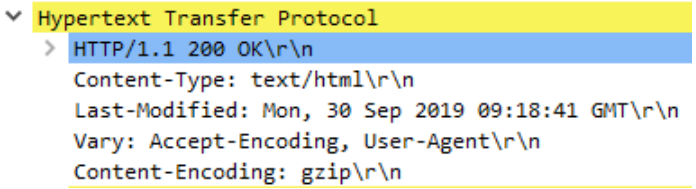
3- Réponses HTTP et représentation des ressources

L'entête des réponses HTTP doit préciser le format des données renvoyés. Celui que nous utiliserons sera JSON.

Réponse HTML : text/html

Réponse XML : application/xml

Réponse JSON : text/json



4-Les codes de statut

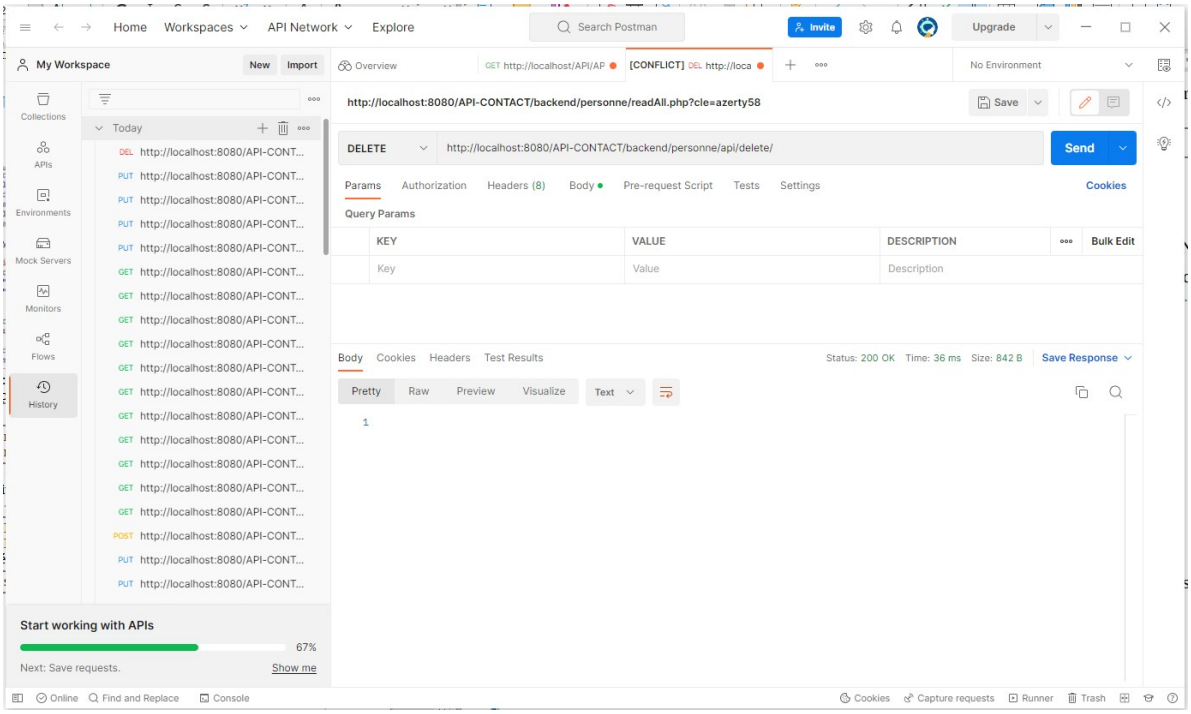
Dans le protocole HTTP, il est prévu de renvoyer un code de statut pour informer le client du résultat de sa requête. Il existe 5 familles :

1xx	Information	Les principaux codes
2xx	Succès	200 : requête traitée avec succès 201 : requête traitée avec succès + création d'un document
3xx	Redirection	301, 302 : ressource déplacée temporairement ou définitivement 304 : cache pas modifié
4xx	Erreur liée au client	404 : Page non trouvée 405 : Méthode de requête non autorisée cas où on s'est trompé de méthode (GET, POST, PUT, DELETE)
5xx	Erreur liée au serveur	503 : serveur non disponible de façon temporaire

Outil

Pour simuler les requêtes GET, il suffit d'appeler l'URL pour afficher les données au format JSON.

Les autres méthodes quant à elles ne peuvent pas être facilement testées. Aussi il est possible de passer par un utilitaire pour faire la simulation. Nous utiliserons POSTMAN qu'il faut l'installer sur son poste de travail (déjà fait au lycée) et disposer d'un compte sur postman.com (vous devez donc vous créer un compte).



Pour en savoir plus sur l'utilisation de Postman, c'est ici.

Json et PHP

Json

Site officiel Json : json.org => pour retrouver toutes les explications d'écriture d'un fichier format Json.

Syntaxe de base :

un objet est caractérisé par des clés auxquelles on associe une valeur sous cette forme :

{"clé" : "valeur"[,
…]}

Le nom de la clé respecte les règles d'écriture d'une variable. Soit par exemple :

```
{
    "pseudo" : "Moi",
    "age" : 19,
    "membre" : true,
    "tempsCnxSemaine" : [60, 55, 86, 15],
    "infos":{
          "login" : "Moi",
          "password" : "azerty58",
          "email" : moi@gmail.com
          }
}
```

Il est possible d'avoir un tableau d'objets [] :

```
[
    {
        "prenom" : "Tom",
        "nom" : "Robert"
    },
    {
        "prenom" : "Hervé",
        "nom" : "Klein"
    }
]
```

Json avec PHP

Comme PHP sera le langage qui va permettre la création du backend, il est nécessaire de connaître les méthodes qui permettent la gestion des données au format Json.

Les fonctions à utiliser :

json_encode() : générer un texte|tableau au format json

```
<?php
    $personnes = array(array('prenom' => 'Paul', 'nom' => 'Eluard'),
        array('prenom' => 'Gustave', 'nom' => 'Flaubert')) ;
    $json = json_encode($personnes) ;
    echo $json ;
?>
```

json_decode(fichier, true) : Récupère une chaîne encodée JSON et la convertit en une valeur de PHP.
le booléen indique comment on veut récupérer les données : true comme tableau associatif,
false sous la forme d'objet.

```php
<?php
    $personne = array('prenom' => 'Paul', 'nom' => 'Eluard');
    $json = json_encode($personne) ;
    $decode = json_decode($json, true)
    echo $decode[json_decode['prenom'];
?>
```

Pour récupération les données d'une API en PHP

```php
<?php
    $url = "https://catfact.ninja/fact";
    $data = file_get_contents($url);
    $decode = json_decode($data, true);
    var_dump($decode);        //Normalement ici on traite l'affichage du résultat au travers d'une vue
?>
```

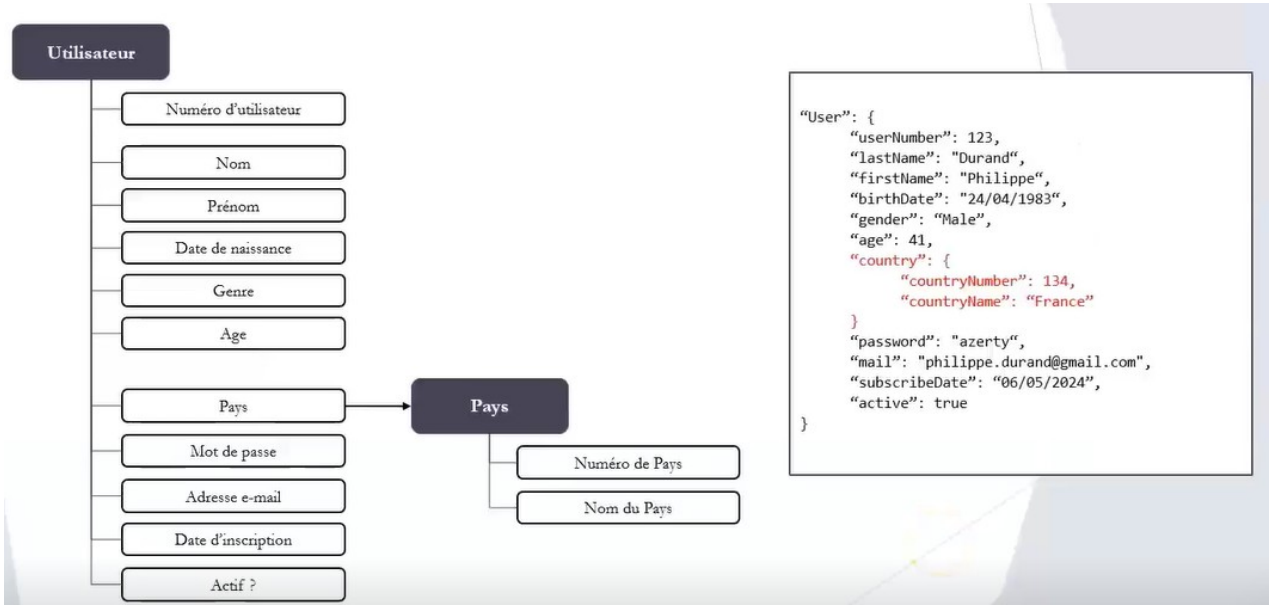
Concevoir une API

Avant de pouvoir coder son API, comme tout développement, cela nécessite de réfléchir à sa conception en se posant la question qui va utiliser mon API (publique, limité à mon entreprise) et les applications qui vont l'utiliser.

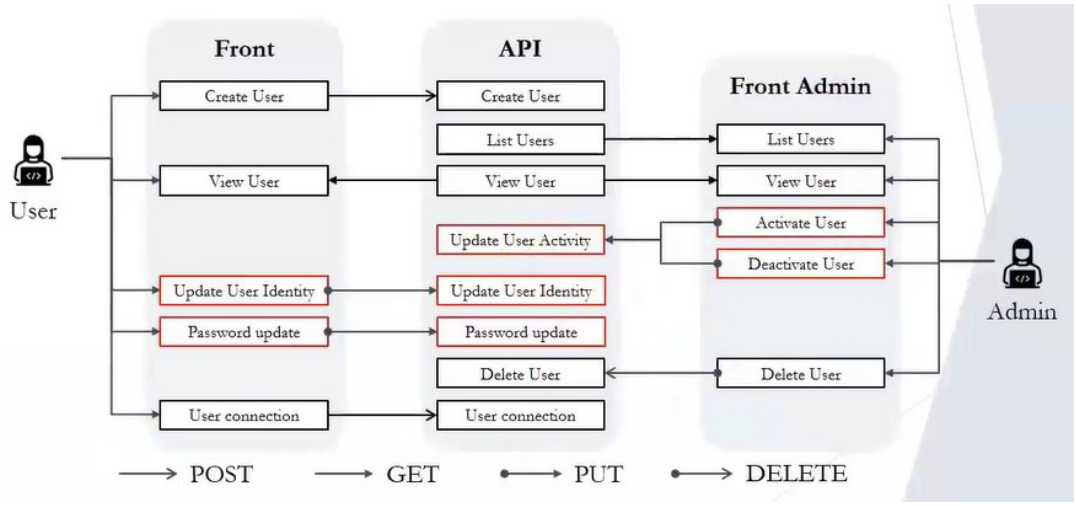
Partons d'un exemple simple, une API qui doit gérer des utilisateurs, mettant en place les opérations CRUD. Maintenant 2 types d'utilisateurs peuvent utiliser l'API, d'un côté l'utilisateur qui est à l'origine de la demande de création de son compte et un administrateur qui pourra activer/désactiver un compte utilisateur et le supprimer si nécessaire.

Pour créer l'API, il faut anticiper les différentes applications qui vont l'utiliser. Ici, il faudra une application pour les utilisateurs et une application pour les administrateurs (1 seule application, gérée avec des droits et des menus différents).

Les informations sur un utilisateur



L'architecture



Les endpoints

C'est la notion la plus structurante d'une API : c'est à dire la requête HTTP.
Le plus simple serait par exemple **GET / users**

Tout endpoint se décompose en 2 :

- La méthode d'appel ici GET
- Le chemin d'accès à la ressource (précise les données à manipuler), ici /users qui correspond à l'objet métier manipulé.

Ici ce qu'on souhaite faire, c'est récupérer les utilisateurs.

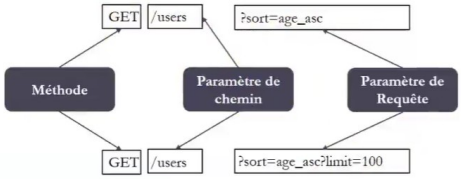
Ce même endpoint pourrait s'écrire : **GET /api/users** ou bien **GET /api/v2/users** ou encore **GET /example.com/api/users**, les seules choses qui nous intéressent sont la méthode et l'objet métier traité.

Autre exemple : **GET/user/{userID}** où ici on enrichit le chemin d'accès en donnant l'identifiant de l'utilisateur comme paramétrage.

Les paramètres

Les paramètres sont toutes les informations entrantes quand on appelle un endpoint. On retrouve :

- **Les paramètres de chemin**, décrits dans le nom du endpoint et permettent au serveur web de déterminer ce que l'on est en train de faire en tant que client de l'API et ce qu'on veut manipuler.
- **Les paramètres de requêtes** pour proposer des options d'appel d'endpoint à nos clients dans la manière de l'appeler.
- **Les paramètres de corps de la requête** qui sont les informations passées dans la requête utilisées pour enregistrer des données dans la BD (informations récupérées d'un formulaire par ex. Ces paramètres sont véhiculés en format Json.).



Les contrôles

Lors d'un échange, l'API va renvoyer un code pour indiquer le statut de la réponse celui-ci est totalement technique. Mais d'autres codes vont être issus du côté fonctionnel notamment pour les règles de gestion métier. Ces contrôles seront exécutés avant l'exécution de la requête, pour s'assurer qu'une saisie utilisateur n'est pas erronnée, que toutes les règles de gestion sont respectées dans les informations indispensables à avoir par exemple ou pour respecter l'intégrité des données en BD, ...

Les codes de contrôle HTTP les plus courantes :

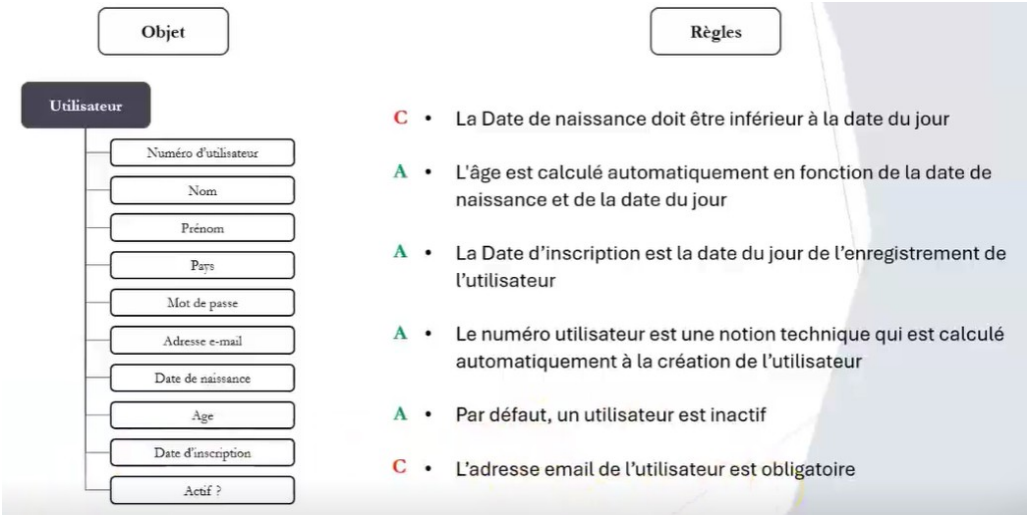
Code	Message	Type	Signification
401	Unauthorized	Technique	La requête nécessite une authentification utilisateur, et celle-ci n'a pas été fournie
400	Bad Request	Technique	Requête non traitée car le serveur ne l'a pas comprises, souvent due à une erreur de syntaxe
503	Service Unavailable	Technique	Serveur incapable de traiter la requête (maintenance ou surcharge)
500	Internal server error	Technique	Serveur a rencontré une condition inattendue qui l'empêche de répondre
404	Not Found	Fonctionnel	Le serveur n'a pas trouvé la ressource correspondant à l'URI demandé
403	Forbidden	Fonctionnel	Le serveur a compris la requête, mais refuse d'autoriser l'action. Contrairement à 401, l'authentification est inutile.
409	Conflict	Fonctionnel	La requête ne peut pas être complétée en raison d'un conflit avec l'état actuel de la ressource.Quelqu'un tente quelque chose qui n'est pas autorisé sur le serveur => la date naissance est > date du jour

Vous en tant que concepteur, ce sont sur les types fonctionnels qu'il faudra agir car cela signifie qu'une personne tente de faire quelque chose sur votre BD qui ne respecte pas les règles de gestion imposées.

Les automatismes

Ce sont des blocs de code qu'on fait exécuter au début du endpoint. Quand on demande un endpoint, cela signifie qu'il y a d'abord les contrôles qui vont s'exécuter et bloquer l'action utilisateur s'il y a un problème. En ce qui concerne les automatismes, c'est plutôt une fois que les données sont enregistrées qu'on va demander à exécuter autre chose dans la BD pour maintenir son intégrité (cela peut être automatisé via les triggers), ou dans d'autres systèmes.

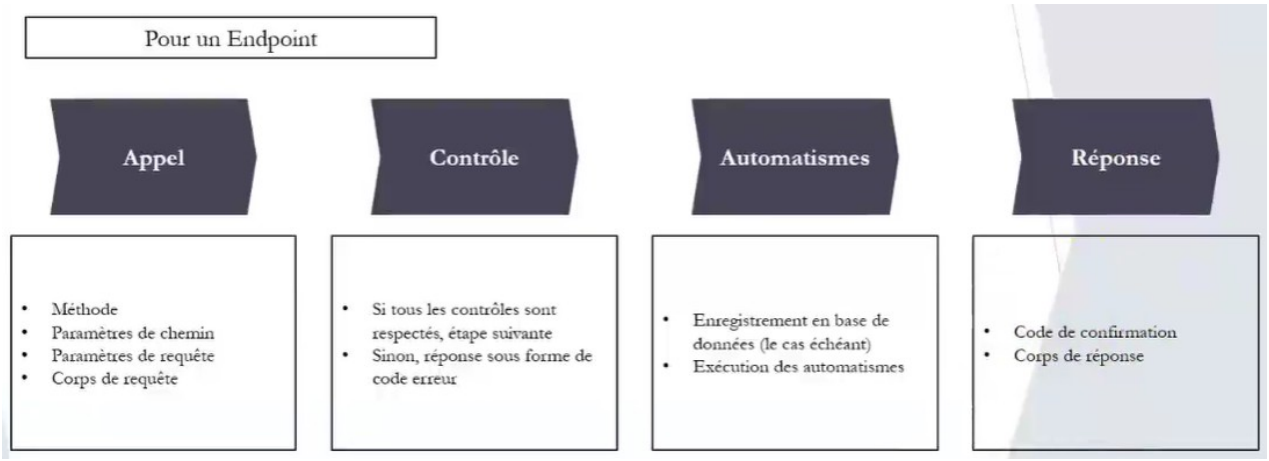
Par exemple pour notre objet Utilisateur, voyons les règles qui pourraient être imposées et définir si la règle est un contrôle ou un automatisme.



Les réponses

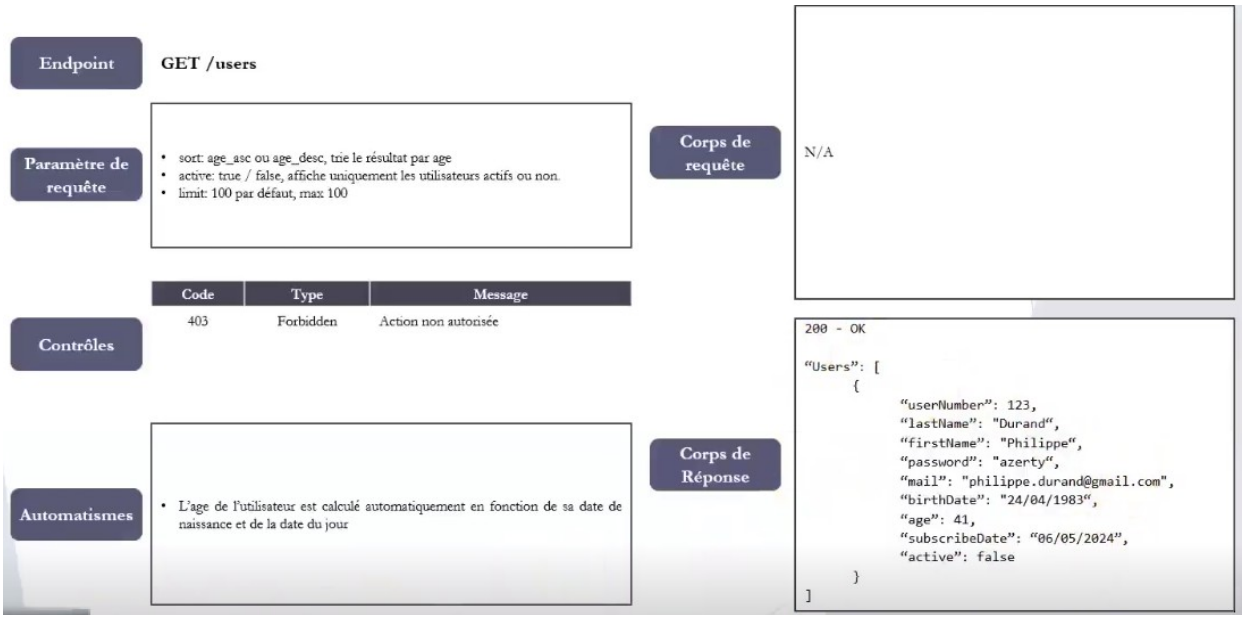
Derniers point important quand on parle API, les réponses. Les erreurs peuvent être considérées comme des réponses, c'est pour cela qu'elles sont accompagnées d'un code, tout comme quand tout c'est bien passé (cf page 3).

Lorsqu'on développe un endpoint (en suivant les caractéristiques d'un endpoint donnés précédemment pour notre cas), il faut définir :

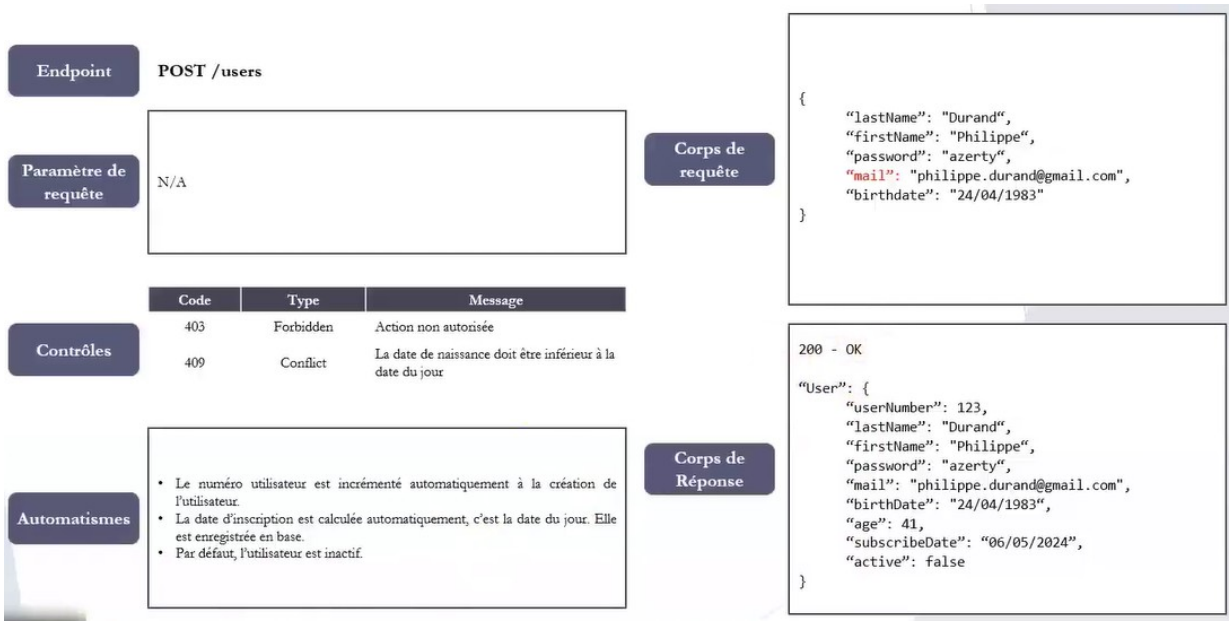


La partie automatisme n'a pas lieu bien entendu quand il s'agit d'une méthode GET, car rien en change en terme d'intégrité des données.

Saine habitude, faire un processus de requêtage pour toutes les méthodes à définir (GET, PUT, POST,…) en utilisant le tableau ci-dessous. Tous les cas ne seront pas énumérés, juste un pour chaque méthode :



Ici il n'y a que les admins autorisés à faire cette action (contrôle : 403)

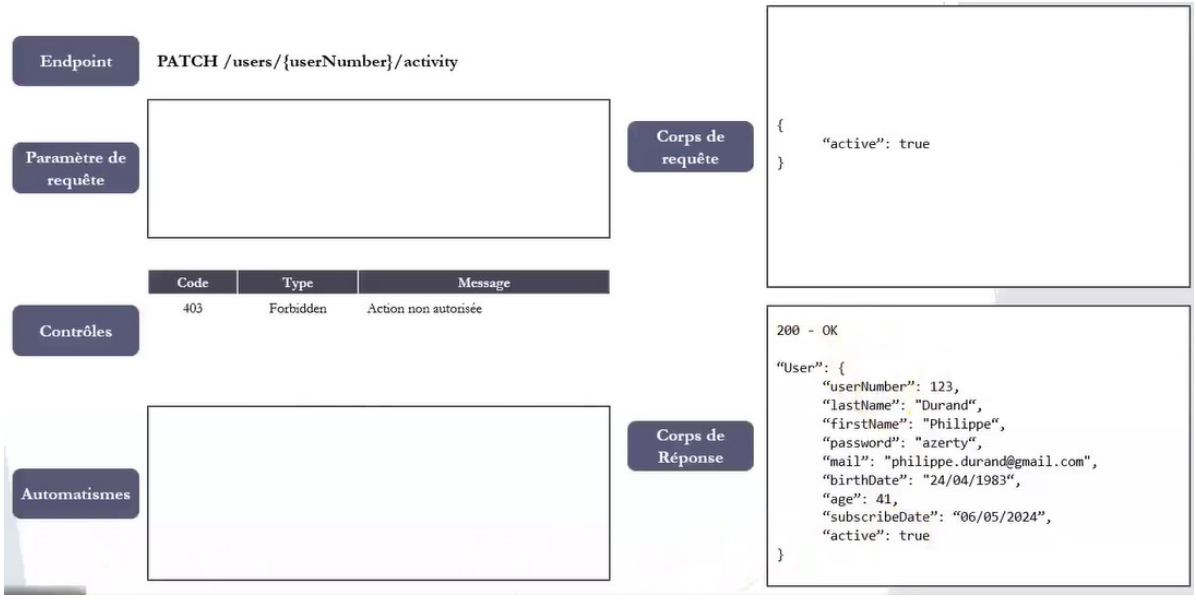


Le post pour la création d'un utilisateur :

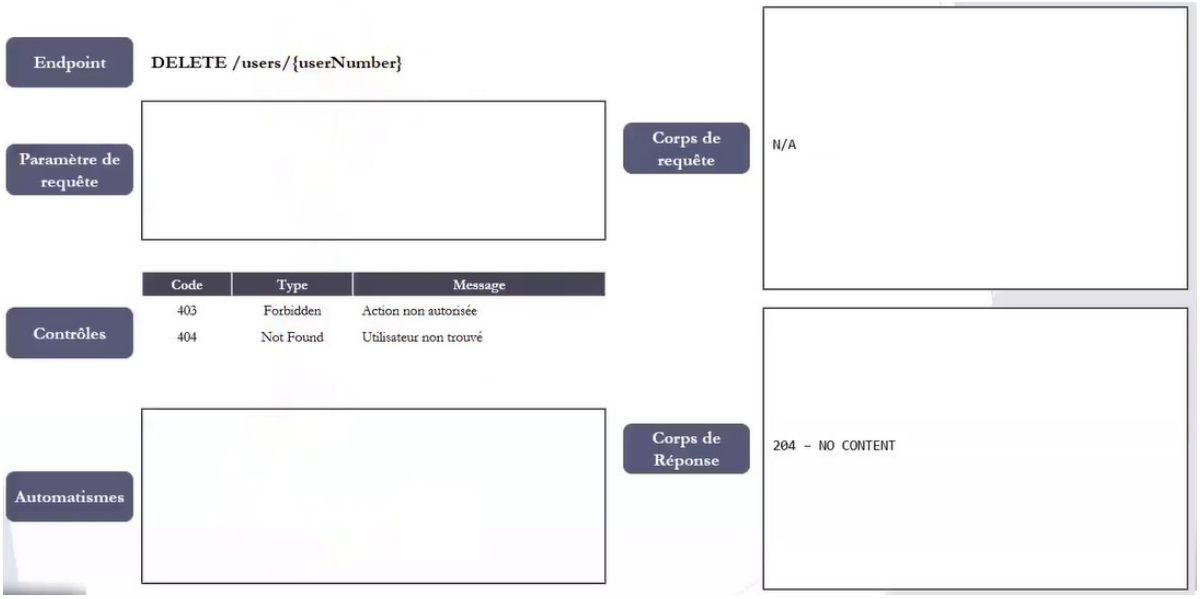
403 car l'administrateur n'a pas le droit de créer un utilisateur, c'est l'utilisateur qui se créé en ligne.

Corps de requête : les informations qu'on veut passer pour l'enregistrement

Il est important de mettre un retour pour afficher les informations sur IHM de l'utilisateur avec toutes les informations le concernant : comme l'age calculé, la définition de son identifiant…



La méthode PATCH, côté administrateur pour activer le compte, donc modifier le champ active.



204 : la requête a été traitée avec succès, mais il n'y a rien à afficher dans le corps de la réponse.

Conclusion

Les API REST sont couramment utilisées, il est important de maîtriser son utilisation et sa création.

En tant que client, vous aurez accès à :

- la spécification de l'API (swagger, OpenAPI Specification, …)
- le document de référence de l'API selon sa complexité

En tant que fournisseur d'API, votre équipe devra produire :

- La spécification de l'API.
- Le document de référence de l'API selon sa complexité.
- Les spécifications du projet (cahier des charges, user story, …).