**API REST ou Restful :**

* Exemple d’API offrant ses données o Les différents types de requêtes (Verbes HTTP) GET, POST, PUT, DELETE
* Les codes HTTP
* Plateforme Apigee, Postman API
* Extension Flask-RESTPlus, librairie requests.

### **QU’EST-CE QU’UNE API ?**

L’API, pour **Application Programming Interface**, est la partie du programme qu’on expose officiellement au monde extérieur pour manipuler celui-ci. L’API est au développeur ce que l’User Interface est à l’utilisateur. Cette dernière permet d’entrer des données et de les récupérer la sortie d’un traitement. Initialement, une API regroupe un ensemble de fonctions ou méthodes, leurs signatures et ordre d’usage pour obtenir un résultat.

La mise en place d’une API permet d’opérer une séparation des responsabilités entre le client et le serveur. Cette séparation permet donc une portabilité et évolutivité grandement améliorées. Chaque composant peut évoluer séparément car il n’y a aucun logique du côté du serveur. Ains on peut imaginer une refonte totale de la charte graphique du site web sans devoir modifier le code côté serveur ou sur les autres clients (mobiles par exemple).

### **API REST OU API SOAP ?**

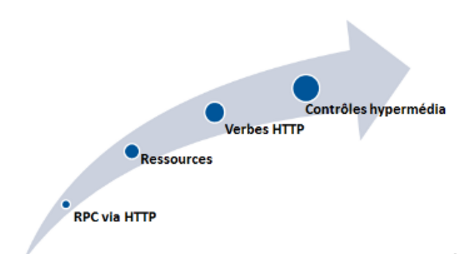
Il existe actuellement deux types d’architecture très utilisées pour les API : Simple Object Access Protocol (SOAP)et Representational State Transfer (REST).

SOAP et REST ​​sont deux solutions permettant à un client d’accéder à des services web. Le choix d'abord peut sembler facile, mais parfois il peut être étonnamment difficile. D’un côté, SOAP, initialement développé par Microsoft, est un protocole d'accès aux services Web qui existe depuis un certain temps. De l’autre, l’architecture REST est la novuelle venue. Elle vise à résoudre certains problèmes rencontrés avec SOAP et donner la possibilité de mettre en place une méthode vraiment simple afin d’accéder à des services web.

Les deux techniques ont des problèmes à prendre en compte au moment de décider quel protocole utiliser. Avant d'aller plus loin, il est important de préciser que même si SOAP et REST présentent des similitudes en utilisant le protocole HTTP, SOAP est un ensemble plus rigide que REST. REST a une architecture qui ne nécessite pas de traitement et qui est naturellement plus flexible. SOAP et REST reposent sur des règles bien établies que tout le monde a accepté de respecter dans l'intérêt de l'échange d'informations.

### **QU’EST-CE QU’UNE API REST ?**

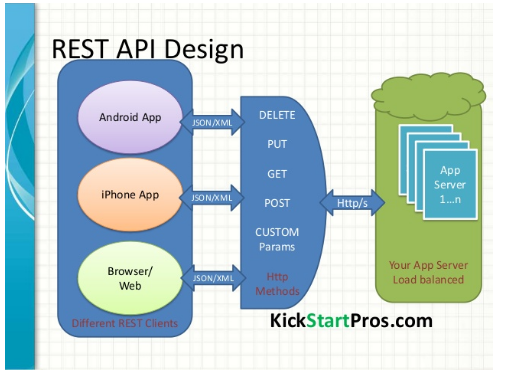
Une API REST se doit d’être sans état ou stateless en anglais. La communication entre le client et le serveur ne doit pas dépendre d’un quelconque contexte provenant du serveur. Ainsi, chaque requête doit contenir l’ensemble des informations nécessaires à son traitement. Cela permet au de traiter indifféremment les requêtes de plusieurs clients via de multiples instances de serveurs.



Selon le modèle de maturité de Richardson il existe quatre grands niveaux d’évaluation d’une API. Au plus on atteint un niveau élevé au plus notre API est considérée comme RESTful et respecte ce qu’on appelle communément les “bonnes pratiques”.

Le niveau 0 (le niveau le plus bas mis en avant par le modèle de Richardson) est une API qui qu’on ne peut pas qualifier d’API REST. Ce niveau s’apparente davantage à ce que l'on peut retrouver dans une API de type SOAP, une architecture un peu démodée, beaucoup moins populaire que l’architecture RESTful. Considéré comme la base minimale, Il s'agit de n'utiliser qu'un seul point d'entrée pour communiquer avec l'API. Une URI unique est mise en place, comme par exemple “/api”, de même qu'une seule méthode HTTP n’est utilisée afin d’effectuer ses demandes, il s’agit de POST. Toutes les requêtes sont de type POST et sont effectués vers la même URI.

Le niveau 1 concerne les différentes ressources et exige en premier lieu que chacune d’elle puisse être distinguée séparément. Cela signifie que l’on ne possède donc plus une unique URI mais une URI pour chaque ressource que l’on souhaite manipuler. Et pour aller plus loin, il est nécessaire de réfléchir dès le début à la manière nous allons mettre en place ces différents URIs. Prenons notre cas pour un exemple concret : avec une API nous permettant de manipuler des athlètes, nous souhaitons mettre en place un CRUD (Create, Read, Update, Delete) soit Créer, Lire, Mettre à jour et Supprimer. Quels sont les points d’entrée qu’il faudra définir afin de pouvoir réaliser ces actions ? Il n’existe pas de solution unique, mais voici une façon de faire qui respecterait le niveau 1 du modèle de Richardson :



* Création/Ajout d’un athlète : /athlète/create
* Pour la lecture d’athlètes, nous aurions l’URI suivante /athlète pour obtenir la liste des athlètes et /athlète/{identifiant-unique} afin d’accéder au profil d’un athlète spécifique par exemple
* Pour la mise à jour des informations d’un athlète cela pourrait être : /athlète/{identifiant-unique}/update
* Pour la suppression, rien de bien compliqué il suffit de changer le verbe de l’URI précédente avec /athlète/{identifiant-unique}/delete

Dernière étape nécessaire afin d’atteindre le 2ème palier, il faut maintenant associer les différentes méthodes HTTP en fonction de l’action que nous souhaitons effectuer grâce à l’API. En reprenant l’exemple de tout à l’heure, il nous suffit d’associer une méthode HTTP (POST, GET, PUT et DELETE) à l’action souhaitée. Ainsi il est toujours possible de réaliser un CRUD et les URIs devraient ressembler à ceci :

* Création/Ajout d’un athlète : POST /athlètes
* Pour la lecture d’athlète GET /athlètes pour obtenir la liste des athlètes et /athlète/{identifiant-unique} afin d’accéder au profil d’un athlète spécifique par exemple.
* Pour la mise à jour des informations d’un athlète cela pourrait être : PUT /athlètes/{identifiant-unique}
* Pour la suppression, DELETE /athlètes/{identifiant-unique}

Les actions ne font plus partie de l'URI, elles sont directement effectuées selon la méthode HTTP utilisée, le protocole est ainsi utilisé à sa pleine capacité. Par ailleurs, dernier point à vérifier et à mettre en place, il s’agit des codes status.

Pour chaque réponse renvoyée par l’API, un code doit être envoyé, ce code correspond à l’état de la requête et dépend de la réussite ou non de celle-ci. Les codes status les plus courants que l’on retrouve généralement sur le web sont :

* 200 OK Tout s'est bien passé
* 201 Created La création de la ressource s'est bien passée (il n’est pas rare que les attributs de la nouvelle ressource soient aussi renvoyées dans la réponse. Dans ce cas, l’URL de cette ressource nouvellement créée est ajouté via un header Location )
* 204 No content Même principe que pour la 201, sauf que cette fois-ci, le contenu de la ressource nouvellement créée ou modifiée n'est pas renvoyée en réponse
* 304 Not modified Le contenu n'a pas été modifié depuis la dernière fois qu'elle a été mise en cache
* 400 Bad request La demande n'a pas pu être traitée correctement
* 401 Unauthorized L'authentification a échoué
* 403 Forbidden L'accès à cette ressource n'est pas autorisé
* 404 Not found La ressource n'existe pas
* 405 Method not allowed La méthode HTTP utilisée n'est pas traitable par l'API
* 406 Not acceptable L’API est dans l’incapacité de fournir le format demandé par les en têtes Accept. Par exemple, le client demande un format (XML par exemple) et l'API n'est pas prévue pour générer du XML
* 500 Server error Le serveur a rencontré un problème.

Il est important de configurer ces statuts et dans le mesure du possible de fournir une explication avec chaque code, de cette façon il est plus facile de résoudre un problème si l’on rencontre un code d’erreur 400 et qu’un message nous explique le problème.

Il existe de nombreux autres critères afin d’établir une API répondant aux normes REST, cependant les points cités ci-dessus sont les plus importants et présentent les avantages de ce type d’API. Le modèle de Richardson met en évidence un troisième niveau d’exigence afin de créer une API parfaitement RESTful, cependant nous avons décidé que dans notre cas, le deuxième niveau serait amplement suffisant et permettrait de travailler avec une API fonctionnelle tout en ayant une architecture soignée. Le but recherché est de pouvoir fournir les données et d’établir le lien entre la logique de présentation se trouvant côté client et nos données se trouvant dans notre base de données.