Introduction à REST et concepts fondamentaux

Les applications web modernes utilisent presque systématiquement des appels à des fonctionnalités distantes. Cette approche permet notamment de pouvoir bénéficier de fonctionnalités riches et réutilisables mises à disposition sur Internet par des éditeurs. Ainsi, intégrer une carte géographique, utiliser des fonctionnalités de paiement en ligne ou encore intégrer les réseaux sociaux dans une application ou un site web est largement facilité.

Les concepts sous-jacents sont ceux des services web. Historiquement articulés autour du langage XML, cette approche s'est simplifiée avec l'arrivée des services en architecture REST, permettant de s'affranchir de la lourdeur d'XML. Les services en architecture REST permettent de mettre à disposition de véritables API utilisables et invocables en HTTP sur Internet.

1. Les concepts de REST

REST est un acronyme signifiant *REpresentational State Transfer*. L'idée est de placer une application dans un état de représentation de ses ressources en fonction de l'action demandée. Cette architecture de construction d'application s'appuie sur le protocole HTTP et sur la notion de ressource.

REST n'est pas à proprement parler un standard dans la mesure où le W3C (*World Wide Web Consortium*) ne le définit pas comme tel. Par contre, il utilise des standards existants du W3C, tels que :

- le protocole HTTP ;
- les URL ;
- les types MIME.

a. Les ressources

REST part du principe qu'Internet est constitué de ressources uniques, chacune d'entre elles étant associée à une URL elle-même unique. Ainsi, à l'URL https://www.meteo.fr/paris se trouve associée une ressource représentant la météo à Paris. Cette ressource peut prendre la forme physique d'une page HTML, d'un script PHP ou bien d'une route Symfony.

b. Le changement d'état d'une ressource

Ainsi définies, les ressources sont sollicitées grâce à des requêtes HTTP. Ces requêtes sont associées à une URL (pour identifier la ressource) ainsi qu'à une **méthode HTTP** (on parle aussi de **verbe HTTP**). La méthode HTTP permet d'identifier l'action que vous souhaitez faire faire à la ressource et donc de changer son état de représentation.

Selon que vous utilisez la méthode **GET** ou bien **POST**, l'état de la ressource n'est pas le même.

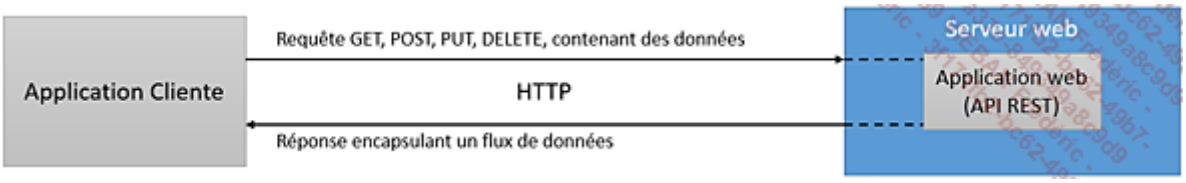
C'est là toute la clé des architectures REST.

2. Architecture et protocole HTTP

Dans le contexte des architectures REST, le protocole HTTP n'est donc pas utilisé pour échanger des pages entre le navigateur et un serveur web, mais pour échanger des données applicatives entre une application cliente (qui pourrait être un navigateur) et une application web hébergée sur un serveur ; cette application sert d'API.

Ces données applicatives peuvent être exprimées dans différents formats (texte brut, XML, JSON…) et peuvent être transmises :

- dans la requête : quand l'application cliente souhaite envoyer des données vers l'API, pour créer une ressource par exemple ;
- dans la réponse : quand l'API renvoie un résultat suite à une invocation de l'application cliente.



D'un point de vue des technologies, l'application cliente peut revêtir plusieurs formes. Il peut s'agir d'une application mobile sur un smartphone ou une tablette, d'une application de bureau déployée sur un poste de travail utilisateur ou bien encore d'un site web distant.

3. Les Single-Page Applications

Une grande majorité des applications et sites web actuels utilisent une approche un peu différente de l'approche MVC évoquée tout au long de cet ouvrage ; ce sont les Single-Page Applications. Le principe repose sur l'usage d'une structure HTML unique pour afficher toutes les informations. Cette structure HTML est dynamiquement modifiée par du code JavaScript en fonction des actions de l'utilisateur. Le changement de page est donc transparent pour l'utilisateur (il ne constate pas de rafraîchissement complet des pages), car seules les données devant être mises à jour sont modifiées dans la structure HTML.

Ces Single-Page Applications utilisent massivement des frameworks basés sur le langage JavaScript afin d'optimiser le développement côté client : on parle de développement **FrontEnd**.

Ainsi toute la logique de modification de cette structure HTML est pilotée par des frameworks tels qu'**Angular**, **VueJS** ou **React**. Ils se chargent également d'apporter des éléments d'interface graphique évolués, simplifiant ainsi la conception des pages.

Afin d'obtenir les données, ces frameworks vont solliciter la plupart du temps une API REST mise à disposition sur un serveur web, comme illustré précédemment, ce sont donc des clients à part entière.

Des composants émettent des requêtes HTTP à destination de l'API et reçoivent des réponses encapsulant les données (souvent en JSON), ces données servant ensuite à mettre à jour la structure HTML de la page.