Présentation des API de service Web REST

Qu' est-ce qu'une API?

Une API permet à un logiciel de communiquer à un autre.

La programmation d'une API va consister à définir comment un logiciel peut contrôler les fonctionnalités qu'elles exposent et les interactions possibles (création, mise à jour ou suppression de données) de manière sécurisée aux autres logiciels ou à un utilisateur.

Voici quelques-uns cas d'utilisation pour les API:

- **Tâches d'automatisation** : création d'un script qui exécute des tâches manuelles automatiquement ;
- **Intégration de données** : permettre à une application de consommer des données fournies par une autre application ou interagir avec cette autre application. Exemple: un site de commerce électronique qui utilise des services de paiement accessibles de' la banque du client via une API REST.
- **Fonctionnalité** : une application peut intégrer les fonctionnalités d'une autre application dans son produit. Exemple: un site web comme Uber qui échangent des données avec Google Maps pour créer et optimiser les itinéraires de voyage. Ces sites peuvent également intégrer les fonctionnalités de Google Maps dans leurs propres applications et sites Web pour présenter des cartes d'itinéraires en temps réel.

API synchrones

Les API sont généralement conçues pour être synchrones lorsque les données de la requête sont facilement disponibles, par exemple lorsque les données sont stockées dans une base de données ou dans une mémoire interne. Le serveur peut immédiatement récupérer ces données et répondre immédiatement.

L'application cliente qui effectue la demande d'API doit attendre la réponse avant d'effectuer des tâches supplémentaires d'exécution de code.

Avantages : l'application reçoit les données immédiatement. **Inconvénient** : si l'API n'est pas correctement conçue, elle sera un goulot d'étranglement car l'application doit attendre la réponse.

API asynchrone

Si la requête d'API prend un certain temps de traitement pour le serveur ou si les données ne sont pas facilement disponibles, il est préférable d'utiliser des API asynchrones.

L'application cliente doit alors être en mesure d'être informer ou de s'informer lorsque le résultat de sa requête API est disponible.

Avantages : l'application client peut de poursuivre l'exécution sans être bloquée pendant le temps nécessaire au serveur pour traiter la demande. L'application client peut avoir de meilleures performances car elle peut être multitâche et faire d'autres requêtes. **Inconvénient** : l'utilisation inutile ou excessive d'appels asynchrones peut avoir l'effet inverse sur les performances.

Les API REST

Les trois types les plus populaires de styles architecturaux API sont RPC, SOAP et REST.

REST ou REprésententational State Transfer (REST) est un style architectural rédigé par l'informaticien américain Roy Thomas Fielding au chapitre 5 de sa thèse de doctorat, Architectural Styles and the Design of Networkbased Software Architectures, en 2000.

Roy Thomas Fielding définit six contraintes appliquées aux éléments de l'architecture:

- **Client-serveur** : le client et le serveur sont c'est à dire qu'il doit être possible de programmer un client pour plusieurs plates-formes et de simplifier les composants côté serveur.
- **Statique** : la demande du client au serveur doivent contenir toutes les informations dont le serveur a besoin pour effectuer la demande. Le serveur ne peut pas contenir d'états de session.
- **Mémoire cache** : Les réponses du serveur indiquent si la réponse est mise en cache ou non en cache. S'il est possible de mettre en cache, le client peut utiliser les données de la réponse pour les demandes ultérieures.
- **Interface uniforme** : l'interface doit respecter quatre principes dans la forme des messages et la réponse du serveur.
- **Système en couches** : le système est composé de différentes couches hiérarchiques dans lesquelles chaque couche fournit des services uniquement à la couche au-dessus. Par conséquent, il consomme les services de la couche ci-dessous.
- **Code à la demande** : la réponse à une requête API peut contenir du code

Quand ces six contraintes sont appliquées il s'agit d'une API RESTful.

API du service web REST

Une API de service Web REST (REST API) est une interface de programmation (API) qui communique via HTTP tout en respectant les principes du style architectural REST.

Comme une API REST communique via HTTP, elle utilise les mêmes concepts que le protocole HTTP :

- Requêtes/réponses HTTP ;
- Verbes HTTP ;
- Codes d'état HTTP ;
- Entêtes/corps http.

Requêtes API REST

Les requêtes d'API REST sont composées de quatre composants principaux:

- **Identificateur de ressources uniformes (URI)** : parfois appelé Uniform Resource Locator (URL), identifie la ressource que le client souhaite manipuler.

Exemple de requête :

- **Méthode http** : utilisation des verbes HTTP standards POST, GET, PUT, PATCH, DEL

Les méthodes HTTP :

Méthode	HTTP	Action	Description
POST	Créer	Créez un nouvel objet ou une nouvelle ressource.	
GET	Lecture	Récupérez les détails des ressources à partir du système.	
PUT	Mettre à jour		
PATCH	Mise à jour partielle	Mettez à jour certains détails à partir d'une ressource existante.	
DEL	Supprimer	une ressource du système.	

- **En-tête** : l'API REST utilise le format d'en-tête HTTP standard formaté en paires nom-valeur séparées par deux points (:), [name]:[value]. Des en-têtes HTTP standard sont utilisables ainsi des en-têtes personnalisés. Il Les en-têtes de requpete (de demande) incluent des informations supplémentaires qui ne sont pas liées au contenu du message.

Exemple d'en-tête de requête fournissant le jeton d'accès d'identification :

Clé	Exemple de valeur	Description
Autorisation	dmFncmFudDp2YWdyYW50	Fournir des informations d'identification pour autoriser la demande

Les en-têtes d'entité sont des informations supplémentaires qui décrivent le contenu du corps du message.

Exemple d'en-tête d'entité précisant le format des données souhaité :

Clé	Exemple de valeur	Description
Type de contenu	application/json	Spécifier le format des données dans le corps

- **Corps** : contient les données relatives à la ressource que le client souhaite manipuler. C'est en général le cas pour les requêtes qui utilisent les méthodes HTTP POST, PUT et PATCH. Selon la méthode HTTP. Si des données sont fournies dans le corps, le type de données doit être spécifié dans l'en-tête à l'aide de la clé Content-Type.

Les réponses API REST

Les réponses API REST sont similaires aux requêtes, mais sont composées de trois composants principaux:

- **État http** : code d'état à 3 chiffres (le premier chiffre indique la catégorie) pour le client si la requête a réussie ou échouée.
- **En-tête** : est facultatif et utilise le format d'en-tête HTTP standard fournir des informations supplémentaires sous la forme de paire nom-valeur séparés par deux points (:), [name]:[value].
- **Corps** : est facultatif et contient les données des demandées en spécifiant le type de données dans l'en-tête à l'aide de la clé Content-Type. En cas d'échec de la requête de l'API REST a échoué, le corps peut fournir des informations supplémentaires sur le problème ou une action qui doit être prise pour que la demande réussisse.

Les cinq catégories de réponse d'API REST :

- **1xx – Informationnel** : informatif, indiquant que le serveur a reçu la demande mais n'a pas fini de la traiter. Le client devrait s'attendre à une réponse complète plus tard. Ces réponses ne contiennent généralement pas de corps.
- **2xx – Succès** : le serveur a reçu et accepté la demande. Quand API synchrones, réponses contiennent les données demandées dans le corps (le cas échéant). Pour les API asynchrones, les réponses ne contiennent généralement pas de corps et le code d'état 2xx est une confirmation que la demande a été reçue mais n'est pas encore satisfaite.
- **3xx - Redirection** : le client a une action supplémentaire à entreprendre pour que la demande soit complétée. La plupart du temps, une URL différente doit être utilisée. Selon la façon dont l'API REST a été invoquée, l'utilisateur peut être automatiquement redirigé sans aucune action manuelle.
- **4xx - Erreur du client** : la requête client contient une erreur (mauvaise syntaxe ; entrée non valide) qui empêche la requête d'être terminée. Le client doit prendre des mesures pour résoudre ces problèmes avant de renvoyer la demande.
- **5xx - Erreur du serveur** : le serveur est incapable de répondre à la demande même si la demande elle-même est valide. Selon le code d'état 5xx particulier, le client peut vouloir réessayer la demande ultérieurement.

Les codes d'état HTTP communs :

Code d'état HTTP	Messages d'état	Description
200	OK	La requête a réussi et contient généralement une charge utile (corps)
201	Créé	La demande a été exécutée et la ressource demandée a été créée
202	Accepté	La demande a été acceptée pour traitement et est en cours
400	Demande incorrecte	La demande ne sera pas traitée en raison d'une erreur avec la demande
401	Non autorisé	La requête n'a pas d'informations d'identification d'authentification valides pour effectuer la demande
403	Interdit	La demande a été comprise mais a été rejetée par le serveur
404	Introuvable	Impossible d'exécuter la demande car le chemin de ressource de la demande n'a pas été trouvé sur le serveur
500	Erreur interne du serveur	La demande ne peut pas être traitée en raison d'une erreur de serveur
503	Service non disponible	La demande ne peut pas être traitée car actuellement le serveur ne peut pas gérer la demande

Utilisation de diagrammes de séquence avec l'API REST

L'interaction avec un service d'API REST particulier consiste en une séquence de demandes. Pour cette raison, les diagrammes de séquence sont fréquemment utilisés pour expliquer la requête/réponse de l'API REST et l'activité asynchrone. <uml>

Requête GET

Client → Serveur*HTTP*S : HTTP/S : créer session\navec accréditation Serveur*HTTP*S → Client : jeton\nd'accréditation Client → Serveur*HTTP*S : HTTP/S : GET\nliste des périphériques Serveur*HTTP*S → Service*API : GET\nliste des périphériques Service*API → BDDConfiguration : Fetch\nliste des périphériques BDDConfiguration → Service*API : Réponse\nliste des périphériques Service*API → Serveur*HTTP*S : Réponse\nliste des périphériques Serveur*HTTP*S → Client : HTTP/S Réponse\nliste des périphériques\nCode réponse 200\nSuccess

Requête POST en asynchrone

Client → Serveur*HTTP*S : HTTP/S : POST\nAjout équipement X Serveur*HTTP*S → Service*API : ADD\nAjout équipement X Service*API → Coeur*app : initiation\najout équipement Coeur*app → Service*API : Réponse\nID de la tâche Service*API → BDDConfiguration : ADD\nAjout équipement X Service_API → Client : HTTP/S : réponse ID de la tâche\nCode réponse 2020 (accepted)

BDDConfiguration → Service*API : \nEcriture faite Service*API → Serveur*HTTP*S : \nEcriture faite\navec succès Serveur*HTTP*S → Client : HTTP/S Réponse\nliste des périphériques </uml>

Authentification à une API REST

Pour des raisons de sécurité, la plupart des API REST nécessitent une authentification afin que les utilisateurs aléatoires ne puissent pas créer, mettre à jour ou supprimer des informations de manière incorrecte ou malveillante, ou accéder à des informations qui ne devraient pas être publiques. La nécessité d'une authentification est le même concept que l'exigence d'un nom d'utilisateur/mot de passe pour accéder à la page d'administration d'une application.

Des API qui ne nécessitent pas d'authentification sont généralement en lecture seule et ne contiennent pas d'informations critiques ou confidentielles.

L'authentification permet de **prouver l'identité** du client.

L'autorisation définit **l'accès** du client à des données.

Mécanismes d'authentification

Les types courants de mécanismes d'authentification incluent Basic, Bearer et API Key.

Authentification de base

Cette authentification utilise le schéma d'authentification HTTP de base standard en transmettant les informations d'identification sous forme de **paires nom d'utilisateur/mot de passe** séparées par un deux-points (:) et encodées à l'aide de Base64.

C'est le mécanisme d'authentification le plus **simple** mais est extrêmement **peu sécurisé** à moins qu'il ne soit jumelé avec des requêtes utilisant HTTPS plutôt que HTTP. Bien que les informations d'identification soient encodées, elles ne sont **pas chiffrées**. Il est simple de décoder les informations d'identification et d'obtenir la paire nom d'utilisateur/mot de passe.

Dans une requête d'API REST, les informations d'authentification de base seront **fournies dans l'en-tête**:

```
Authorization: Basic <username>:<password>
```

Authentification du porteur

Ce mécanisme d'authentification appelé aussi **authentification au jeton**, utilise le schéma d'authentification HTTP au porteur standard. Il est **plus sécurisé** que l'authentification de base et est généralement utilisé avec **OAuth** et **Single Sign-On (SSO)**. L'authentification au porteur utilise un jeton porteur, qui est une chaîne générée par un **serveur d'authentification externe** tel qu'un service d'identité (ID).

Tout comme l'authentification de base, l'authentification au porteur doit être utilisée avec HTTPS.

Dans une requête d'API REST, les informations d'authentification du porteur seront fournies dans l'en-tête :

```
Authorization: Bearer <bearer token>
```

Clé API

Une **clé API** ou jeton API, est une **chaîne alphanumérique unique générée par le serveur** et affectée à un utilisateur. Pour obtenir une clé API unique, l'utilisateur se connecte généralement à un portail à l'aide de ses informations d'identification. Cette clé est généralement assignée une fois et ne sera pas régénérée. Toutes les requêtes d'API REST pour cet utilisateur doivent fournir la clé d'API assignée comme forme d'authentification.

Tout comme avec les autres types d'authentification, les clés API ne sont sécurisées que lorsqu'elles sont utilisées avec HTTPS.

Les clés API sont destinées à être un mécanisme d'authentification, mais sont généralement utilisées à mauvais escient comme mécanisme d'autorisation.

Les deux types de clés API sont publiques et privées.

- Une clé d'API publique peut être partagée et permet à cet utilisateur d'accéder à un sous-ensemble de données et d'API.
- une clé privée ne doit pas être partagée, car elle est similaire à votre nom d'utilisateur et votre mot de passe.

La plupart des clés d'API n'expirent pas, et à moins que la clé ne puisse être révoquée ou régénérée, si elle est distribuée ou compromise, toute personne disposant de cette clé peut accéder indéfiniment au système comme vous.

Une requête API REST peut fournir une clé API de différentes manières:

```
Query string: Recommandé uniquement pour les clés API publiques
```

- Header: Utilise la clé Autorisation ou une clé personnalisée <code> Authorization: <API Key> ou Authorization: APIkey <API Key> ou APIkey: <API Key> </code>
- Body data: Utilise une clé unique comme identifiant <code> Content-Type: application/json </code>
- Cookie: Utilise une clé unique comme identifiant <code> Cookie: API*KEY=<API Key> </code> </WRAP> ==== Retour Activité A8 ==== * *A8 - La gestion des configurations avec GLPI*