Architecture Logicielle

Introduction à REST

Faire communiquer des programmes entre différentes machines

- Socket
- RPC
- RMI
- CORBA
- ...

Problème de compatibilité, de passage à l'échelle, ...

Faire communiquer des programmes à travers des services web

- XML-RPC
- BPEL
- SOAP, WSDL, UDDI
- ...

Bilan de ces services web

- Assez complexe
- Interopérabilité limitée
- Reutilisabilité limitée, couplage fort entre client et serveur
- Difficulté pour passer à l'échelle

REST :REpresentational State Transfer

- Architecture logicielle pour définir des services web
- Les principes de REST ont été théorisés par Roy Fielding dans sa thèse en 2000
- `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`

Intentions

- S'inspirer des bonnes pratiques qui ont fait le succès du Web
- Généralisation d'un style d'architecture
- Définition de contraintes pour respecter ce style
- Pas une nouvelle technologie, pas un nouveau langage

REST se distingue par les avantages que cette architecture offre :

Caractéristiques

- Simplicité des serveurs
- Capacité de montée en charge des serveurs
- Équilibrage des charges
- Bonne gestion du cache
- Décomposer des services complexes en plusieurs services simples qui communiquent entre eux

Pour aboutir à ces caractéristiques, REST est définie par plusieurs grands principes

Principes

- Séparation claire entre Client et Serveur
- Serveur sans état
- Cache
- Interface uniforme
- Système en couches

Séparation claire entre Client et Serveur

Le rôle du client et du serveur sont clairement réparties. On diminue ainsi le couplage entre ces deux éléments :

- La portabilité est améliorée en séparant l'interface utilisateur du stockage des données
- La capacité de passage à l'échelle est améliorée en simplifiant le rôle du serveur
- Indépendance : les différents composants peuvent évoluer de manière indépendante

Serveur sans état

- Le serveur ne conserve pas l'état de la communication avec le client
- Le client doit donc envoyer à chaque requête toutes les informations nécessaires. L'état de la session est conservé entièrement par le client

Avantages

- Visibilité : le serveur a juste besoin d'examiner la requête
- Fiabilité : facilite le retour à l'état normal en cas d'échec
- Passage à l'échelle : le serveur est simplifié, besoin de moins de ressource

Comment gérer l'authentification ?

Cache

Les réponses du serveur peuvent être marquées comme étant possible à mettre dans le cache. Si une réponse est "cachable", le client peut la réutiliser au lieu de refaire une requête

- performance : réduit la latence
- élimine des interactions

Risque de perte de fiabilité si les données deviennent obsolètes

Interface uniforme

Point central de l'architecture REST : les différents composants doivent utiliser une interface simple, bien définie, standardisée

- Identification des ressources dans les requêtes : chaque ressource est identifiée dans les uri par exemple
- Manipulation des ressources par des représentations : les ressources peuvent être distinctes des représentations envoyées au client (JSON,HTML,XML)
- Message auto descriptif
- *Hypermedia As The Engine of Application State* : le client n'a pas besoin de connaissance au préalable pour intéragir avec le serveur. Il utilise les hyperliens pour découvrir les actions possibles et ressources du serveur

Système en couches

Il peut exister des serveurs intermédiaires. Le client ne peut pas distinguer le serveur final d'un serveur intermédiaire

- Équilibrage des charges
- Cache partagée
- Sécurité

Peut poser des problèmes de performances

Code à la demande (facultatif)

Les serveurs peuvent transférer du code au client (par exemples scripts Javascript)

Conclusion

- Simplification des services web
- Inspiré des bonnes pratiques du Web
- Défini par un ensemble de règles

Comment mettre en pratique ces règles ?