Cours Inf3522 – Développement d'Applications N–tiers

Cours Inf3523 – Architecture et Génie des Logiciels

Cours_2 : Architecture Web + API REST

REST signifie Representational State Transfer. C'est un style d'architecture logicielle pour implémenter des services web. Les services web implémentés en utilisant le style architectural REST sont connus sous le nom de services web RESTful

Qu'est-ce qu'une API REST ?

Une API REST est une implémentation d'API qui respecte les contraintes architecturales REST. Elle agit comme une interface. La communication entre le client et le serveur se fait via HTTP. Une API REST utilise les méthodologies HTTP pour établir la communication entre le client et le serveur. REST permet également aux serveurs de mettre en cache la réponse, ce qui améliore les performances de l'application.

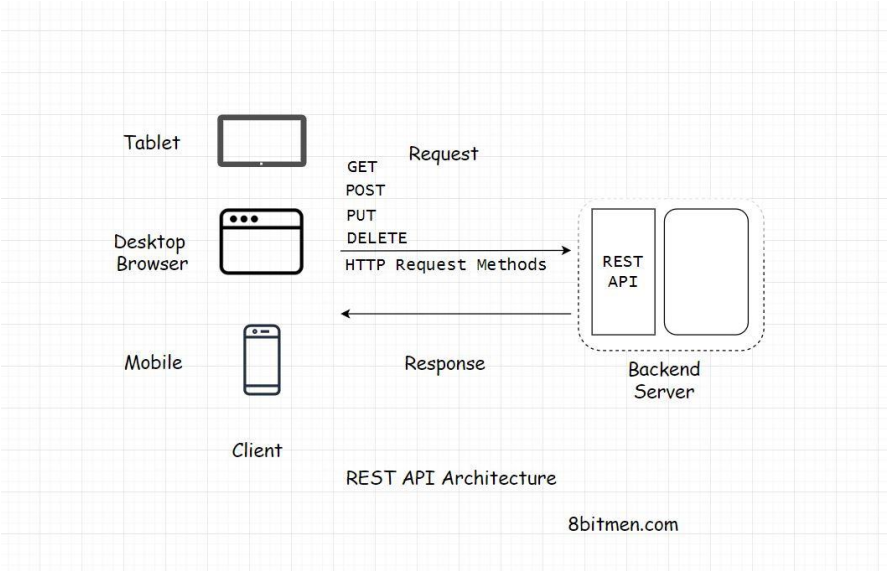

Illustration 1.12 : Architecture de l'API REST

La communication entre le client et le serveur est un processus sans état. Cela signifie que chaque communication entre le client et le serveur est comme une nouvelle communication.

Il n'y a pas d'informations ou de mémoire transportées à partir des communications précédentes. Donc, chaque fois qu'un client interagit avec le backend, le client doit également envoyer les informations d'authentification. Cela permet au backend de déterminer si le client est autorisé à accéder aux données ou non.

Lors de la mise en œuvre d'une API REST, le client communique avec les points de terminaison du backend. Cela dissocie complètement le code backend et le code client.

Point de terminaison REST

Un point de terminaison d'API/REST/Backend est l'URL du service que le client peut utiliser. Par exemple, https://myservice.com/users/{username} est un point de terminaison backend pour récupérer les détails d'un utilisateur particulier à partir du service.

Le service basé sur REST exposera cette URL à tous ses clients pour récupérer les détails de l'utilisateur en utilisant l'URL indiquée ci-dessus.

Dissocier les clients et le service backend

Avec la disponibilité des points de terminaison, le service backend n'a pas à se soucier de l'implémentation du client. Il appelle simplement ses clients multiples et leur dit : « *Hé les gars ! Voici l'adresse URL de la ressource/information dont vous avez besoin. Allez-y quand vous en avez besoin. Tout client ayant l'autorisation requise pour accéder à une ressource peut y accéder* ».

Avec la mise en œuvre de REST, les développeurs peuvent avoir différentes implémentations pour différents clients, en tirant parti de différentes technologies avec des bases de code séparées. Différents clients accédant à une API REST commune pourraient être un navigateur mobile, un navigateur web, une tablette ou un outil de test d'API comme Postman. L'introduction de nouveaux types de clients ou la modification du code client n'affecte pas la fonctionnalité du service backend.

Cela signifie que les clients et le service backend sont dissociés.

Développement d'applications avant l'API REST

Avant que les interfaces d'API basées sur REST ne deviennent courantes dans l'industrie, nous avons souvent couplé étroitement le code backend avec le client. Java Server Pages (JSP) en est un exemple.

Nous mettions toujours la logique métier dans les balises JSP. Cela rendait la révision du code (refactoring) et l'ajout de nouvelles fonctionnalités difficiles car la logique métier était répartie sur différentes couches.

De plus, à l'arrière-plan, nous devions écrire du code/classes distinct pour gérer les requêtes de différents types de clients. Nous avions besoin d'un servlet distinct pour gérer les requêtes d'un client mobile et un autre pour un client web.

Développement d'applications avec l'API REST

Après la mise en œuvre des API REST, les développeurs backend n'avaient plus besoin de se soucier du type de client. Tout ce que les développeurs avaient à faire était de fournir les points de terminaison de service aux clients. Ces derniers recevaient la réponse dans un format de transport de données standard comme JSON. Il était alors de la responsabilité des clients de parser et d'afficher les données de réponse.

Cela a réduit beaucoup de travail inutile pour les développeurs backend. De plus, l'ajout de nouveaux clients est devenu beaucoup plus facile. Maintenant, avec REST, nous pouvons introduire n'importe quel nombre de nouveaux clients sans avoir à nous soucier de la mise en œuvre du backend.

Dans le paysage actuel du développement d'applications, il n'y a pratiquement aucun service en ligne implémenté sans une API REST. Vous voulez accéder aux données publiques d'un réseau social ? Utilisez simplement leur API REST.

Passerelle API REST (REST API Gateway)

L'API REST agit comme une passerelle ou un point d'entrée unique dans le système. Elle encapsule la logique métier et gère toutes les demandes du client, s'occupant de l'autorisation, de l'authentification, de la désinfection des données d'entrée et d'autres tâches nécessaires avant de donner accès aux ressources de l'application.

Maintenant, nous connaissons l'architecture client-serveur et ce qu'est une API REST. Elle agit comme une interface, et la communication entre le client et le serveur se fait via HTTP.

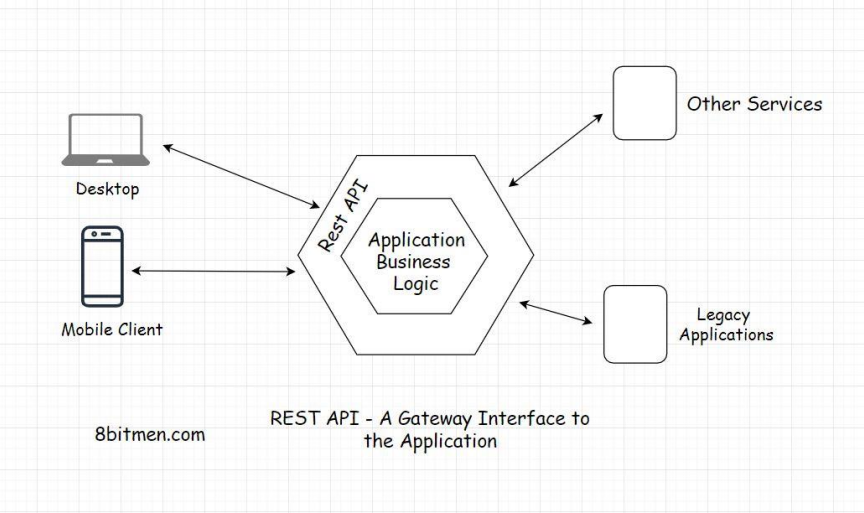

Illustration 1.13: REST API Gateway

Maintenant, examinons le mécanisme de communication Push et Pull basé sur HTTP.

HTTP Pull et Push - Introduction

Dans cette partie, vous aurez un aperçu du mécanisme HTTP Push et Pull. Nous savons que la plupart de la communication sur le web se fait via HTTP, surtout lorsqu'il s'agit de l'architecture client-serveur.

Il existe deux modes de transfert de données entre le client et le serveur: HTTP PUSH et HTTP PULL.

HTTP PULL

Comme je l'ai indiqué plus tôt, pour chaque réponse, il doit y avoir une requête au préalable. Le client envoie la requête et le serveur répond avec les données. C'est le mode de communication HTTP par défaut, appelé le mécanisme HTTP PULL.

Le client extrait les données du serveur chaque fois que cela est nécessaire. Il continue de le faire à chaque fois pour récupérer les dernières données.

Il est important de noter ici que chaque requête au serveur et la réponse à celle-ci consomme de la bande passante. Une requête au serveur peut coûter de l'argent à l'entreprise et peut ajouter de la charge sur le serveur.

Que se passe-t-il s'il n'y a pas de données mises à jour disponibles sur le serveur chaque fois que le client envoie une requête? Le client ne le sait pas, il continuerait donc naturellement à envoyer les requêtes au serveur encore et encore. Ce n'est pas idéal et c'est une perte de ressources. Les demandes excessives des clients ont le potentiel de faire tomber le serveur.

HTTP PUSH

Pour résoudre ce problème, nous avons le mécanisme basé sur HTTP PUSH. Dans ce mécanisme, le client envoie la requête pour obtenir certaines informations auprès du serveur une seule fois. Après la première demande, le serveur continue d'envoyer les nouvelles mises à jour vers le client chaque fois qu'elles sont disponibles.

Le client n'a pas à se soucier d'envoyer des demandes supplémentaires au serveur pour obtenir des données. Cela permet d'économiser beaucoup de bande passante réseau et de réduire considérablement la charge sur le serveur.

Cela est également connu sous le nom de rappel (callback). Le client appelle le serveur pour obtenir des informations. Le serveur répond : « Eh bien !! Je n'ai pas les informations pour le moment, mais je vous rappellerai dès qu'elles seront disponibles ».

Un exemple très courant de cela est les notifications utilisateur. Nous les avons dans presque toutes les applications web aujourd'hui. Nous sommes avertis chaque fois qu'un événement se produit sur le backend.

Les clients utilisent Asynchronous JavaScript & XML (AJAX) pour envoyer des requêtes au serveur dans les mécanismes HTTP PULL et HTTP PUSH.

Il existe plusieurs technologies impliquées dans le mécanisme basé sur HTTP PUSH, telles que :

- Interrogation en attente prolongée d'AJAX
- Sockets web
- Source d'événements HTML5
- Files d'attente de messages
- Diffusion en continu via http (Streaming over http)

Nous allons les examiner en détail dans le cours suivant.