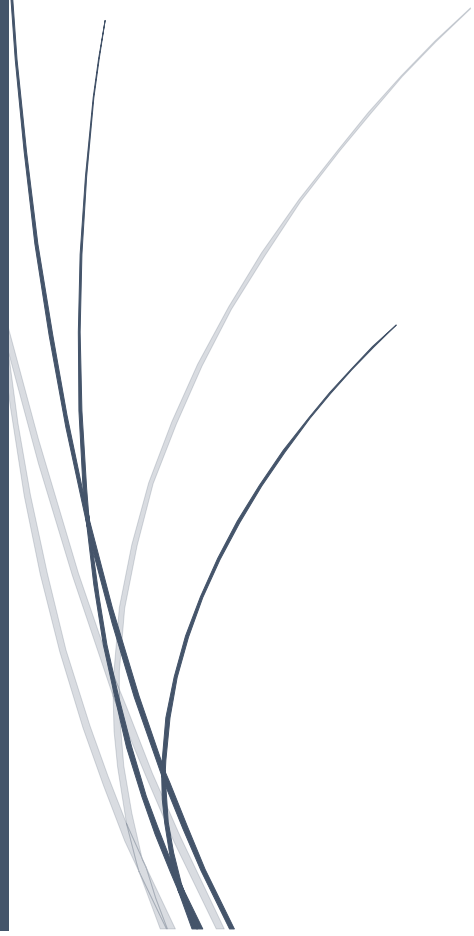
14/11/2019

SYM Labo2

Loris Crüll, Julien Rod, Yoann Rohrbasser



4.1 Traitement des erreurs

Si le serveur ne répond pas, le SymComManager resterait bloqué lorsqu'il essaie de recevoir la réponse. Cela pourrait être un problème si celle-ci n'arrive jamais (perte de message). Pour régler ce problème, il faudrait plutôt ouvrir un nouveau thread pour le SymComManager et rajouter un nouveau thread timeout, qui attend x seconde, regarde si le SymComManager est arrêté et si ce n'est pas le cas, l'arrête et affiche une erreur "serveur indisponible".

Dans le cas où nous recevons un message d'erreur, ce dernier ne ferait qu'afficher ce qu'il reçoit. (Il n'y a aucun contrôle sur le code de réponse). Nous pouvons simplement vérifier le code réponse du serveur via un connection.getResponseCode() puis envoyer un message différent selon si elle désigne une erreur ou si la réponse du serveur est correcte.

4.2 Authentification

Oui, c'est possible. Il suffit de rajouter un header Authorization contenant nos identifiants (en admettant que nous en ayons), il faut aussi que l'application côté serveur permette ce genre d'authentification. Pour les requêtes différées, cela est aussi possible, mais il faut faire en sorte que l'utilisateur n'ait pas à rentrer ses identifiants quand le serveur devient disponible.

4.3 Threads concurrents

Un problème de concurrence. Il faut faire en sorte que les processus soient synchronisés de sortes à ce que nous ne perdions pas de données ni lors de l'envoi (qu'une ancienne préparation se fasse écraser par une nouvelle avant qu'elle ne soit envoyée), ni lors de la réception (qu'une donnée reçue se fasse écraser par une nouvelle avant qu'elle ne soit traitée).

4.4 Ecriture différée

Effectuer une connexion par transmission différée

En effectuant la connexion en différée et pas les messages on limite le nombre de threads en attente active (si l'on assume que le thread de connexion se charge de réveiller les threads de message) et fait en sorte qu'il n'y a que besoin d'une boucle qui ping la connexion à la fois.

Multiplexer les connexions

En multiplexant les connexions on réduit le nombre de connexions ouvertes à une. Cependant il faut implémenter le multiplexeur capable de rediriger les messages là où ils doivent l'être.

4.5 Transmission d'objets

4.5.a

REST/JSON est moins sécurisé et donc serait vulnérable à des attaques, si le code derrière n'est pas sécurisé. En revanche, sa syntaxe est plus simple et fournit de meilleur support pour les navigateurs comme Chrome ou Firefox.

4.5.b

Les Protocol Buffers sont compatibles avec HTTP. Le Protocol Buffer pourrait sérialiser un objet en une requête http pour l'envoyer.

4.5.c

Il faudrait être capable de retirer une plus petite portion de données à la fois. Récupérer trop de données impliquerait beaucoup de travail de la part du mobile pour faire la pagination. Il vaudrait mieux fixer une limite et un offset pour les limiter. Si les utilisateurs veulent en avoir plus on peut ainsi refaire des requêtes au serveur pendant que l'utilisateur scrolle.

4.6 Transmission compressée

Texte	Normal	Compressée
Test	129	309
Lorem Ipsum (premier paragraphe)	230	287
Lorem Ipsum (entier)	Crash	348
{"name":"John Doe","phone":"+4766666666"}	360	270

On constate qu'après une certaine taille la version sans compression plante alors que le format compressé ne plante pas. On constate aussi que pour du texte simple, les performances sont meilleures sans compression. Toutefois, plus le texte est long plus le temps d'attente entre l'envoi de la requête et le temps de réception de la réponse sera long (sans parler des problèmes d'erreurs si le texte est trop long). Dans la version compressée, on constate que le temps d'envoi et de réception est stable quelle que soit la longueur du texte. Il n'y a pas non plus de différences significatives entre le fait que le message envoyé soit du texte simple ou des objets sérialisés lors d'envois compressés.