SYM Labo 2

Auteurs: Adrien Allemand, James Smith et Loyse Krug

Transmission différée

```
Dans le cadre de cette manipulation vous pouvez rester sur une solution simple « en
mémoire », vous indiquerez toutefois dans votre rapport les limitations de cette façon
de faire et proposerez des outils et techniques mieux adaptés (sans forcément réaliser
l'implémentation).
```

Vu que c'est stoqué en mémoire, si on quitte l'application les requêtes en attentes seront perdu. Pour se faire on pourrait enregister les requêtes sur la carte SD lorsque l'application est quitté et qu'on lancement de l'application, on les relances.

Reponse question

1 Traitement des erreurs

```
Les interfaces AsyncSendRequest et CommunicationEventListener utilisées au point 3.1
restent très (et certainement trop) simples pour être utilisables dans une vraie
application : que se passe-t-il si le serveur n'est pas joignable dans l'immédiat ou
s'il retourne un code HTTP d'erreur ? Veuillez proposer une nouvelle version, mieux
adaptée, de ces deux interfaces pour vous aider à illustrer votre réponse.
```

Si une erreur se produit durant la communication avec le serveur, il y a une levée d'exception. Celle-ci est ensuite géré par le thread AsynchTask et gère le problème, l'affiche, l'ignore etc.

Actuellement, l'erreur de connextion est traîtée dans chaque service séparément. Il serait plus propre d'ajouter une fonction qui traîte les cas d'erreur à l'interface CommunicationEventListener.

2 Authentification

```
Si une authentification par le serveur est requise, peut-on utiliser un protocole
asynchrone ? Quelles seraient les restrictions ? Peut-on utiliser une transmission
différée ?
```

Non, on ne veut pas laisser l'utilisateur accéder à l'application alors qu'on a pas la validation de son identité. Tant que l'application nécessite une authentification serveur, il faut attendre la réponse du serveur avant de pouvoir continuer. Une transmission différée n'est pas valide pour les mêmes raisons.

La seul raison pour laquelle une authentification asynchrone serait possible est si l'application possèdes des foncitonnalités ne nécessitant pas d'autorisation et que pendant la durée de l'authentification, on peut laisser l'utilisateur utiliser les fonctionnalités nécessaitant pas d'autorisation uniquement.

3 Threads concurrents

```
Lors de l'utilisation de protocoles asynchrones, c'est généralement deux threads
différents qui se préoccupent de la préparation, de l'envoi, de la réception et du
traitement des données. Quels problèmes cela peut-il poser ?
```

Problème de concurence. Il peut y avoir plusieurs envoie avant meme de recevoir la première reponse.

Donc on peut avoir l'affichage de la modification de la vue avec les données de la première requête alors qu'une deuxième a été saisi. Cela peut causé des confusions chez l'utilisateur.

4 Ecriture différée

```
Lorsque l'on implémente l'écriture différée, il arrive que l'on ait soudainement
plusieurs transmissions en attente qui deviennent possibles simultanément. Comment
implémenter proprement cette situation (sans réalisation pratique) ? Voici deux
possibilités :
- Effectuer une connexion par transmission différée
- Multiplexer toutes les connexions vers un même serveur en une seule connexion de
    transport.Dans ce dernier cas, comment implémenter le protocole applicatif, quels
    avantages peut-on espérer de ce multiplexage, et surtout, comment doit-on planifier
    les réponses du serveur lorsque ces dernières s'avèrent nécessaires ?

Comparer les deux techniques (et éventuellement d'autres que vous pourriez imaginer) et
discuter des avantages et inconvénients respectifs.
```

Mise en situation: Un utilisateur créé un poste sur notre applications qui est un réseau social. Celui ci ce dit qu'il a fait une erreur et le modifie. Et quelque seconde plus tard il se rend compte que les informations qu'il a publié sont totalement fausse et décide de le supprimer. Si l'utilisateur à un problème de connection à ce moment et que les requêtes sont en attente, il peut se confronter à des problèmes.

- Transmission différée.
    - Envoyer requête par requête permet d'alléger le poids des requêtes et en cas d'erreur, evite de devoir renvoyer toutes les requête.
    - En cas de faible connection, malgrès un envoie plus conséquent de donnée (duplication des headers etc), les premières requête peuvent être résolu et le résultat renvoyé. Ce qui peut donné un sentiment d'avancement à l'utilisateur contrairement à la prochaine solution ou tous ce passe en une fois. (La transmission différé réduit les risques d'avoir des timout error.)
    - Mais nous n'avons aucune garantie qu'une requête arrivera avant une autre au serveur.
        - Dans notre cas sela pourrait poser problème si la suppression arrive avant la création ou que la modification arrive après la suppression.
- Multiplexer
    - En cas d'erreur, nous sommes obligés de tous renvoyé, ce qui peut être couteux
    - Un long envoie (moins lourd au niveau taille total à prioris que une transmission différée) mais l'utilisateurs doit attendre que tous soit envoyé et traité avant de voir quelque chose.
    - Résoud le problème de l'ordre d'arrivé des requêtes.

5 Transmission d’objet

```
Quel inconvénient y a-t-il à utiliser une infrastructure de type REST/JSON n'offrant aucun service de validation (DTD, XML-schéma, WSDL) par rapport à une infrastructure comme SOAP offrant ces possibilités ?
```

On peut plus facilement faire des erreurs, moins réutilisable. Nécessite de faire des tests des saisis de l'utilisateur. On ne peut pas juste le faire valdier par un schema comme on pourrait le faire en XML.

```
Est-ce qu'il y a en revanche des avantages que vous pouvez citer ?
```

Plus lisible, plus facile à implémenter. Très pratique couplé à des db comme mango (Schemaless). Laisse plus de flexibilité.

```
L'utilisation d'un mécanisme comme Protocol Buffers 8 est-elle compatible avec une architecture basée sur HTTP ? Veuillez discuter des éventuelles avantages ou limitations par rapport à un protocole basé sur JSON ou XML ?
```

Oui, Protocole Buffer est compatile avec une architecture basé sur HTTP. C'est flexible, léger, et permet une validation comme la DTD de XML. Mais dès que les données sont sérialisées, elles sont transformée en binaire et donc illisible à l'oeil humain. Peut être problèmatique pour des phases de debug.

```
Par rapport à l'API GraphQL mise à disposition pour ce laboratoire. Avez-vous constaté des points qui pourraient être améliorés pour une utilisation mobile ? Veuillez en discuter, vous pouvez élargir votre réflexion à une problématique plus large que la manipulation effectuée.
```

Il n'y a pas de système de pagination mise en place. Peut poser problème si un utilisateur à 100'000 posts. On doit attendre que tous soit chargé pour commencé à les afficher. Et l'utilisateur va surement lire les 10 premier avant de changer de fonctionnalité. Une pagination permetterais de faire des requêtes plus courte et rapide mais plus nombreuse.

6 Transmission compressée

```
Quel gain peut-on constater en moyenne sur des fichiers texte (xml et json sont aussi du texte) en utilisant de la compression du point 3.4 ? Vous comparerez vos résultats par rapport au gain théorique d'une compression DEFLATE, vous enverrez aussi plusieurs tailles de contenu pour comparer.
```

Taille de la requête, nombre de caractères	temps requête simple [ms]	temps requête compressée [ms]	taille de la chaine compressée	rapport de taille compressée/taille non compressée [%]
1	182	161	3	300%
50	157	190	52	104%
500	100	193	402	80%
5000	147	186	3'772	75%
50'000	313	211	37'591	75%
500'000	timeout	281	75'186	15%

Pour comparer le temps d'envoi des requêtes entre des requêtes simples et des requêtes compressées, nous avons ajouté un timer avant et après chacune des deux requêtes et envoyé des requêtes de taille variable. Pour être plus juste dans nos comparaisons, nous avons ajouté l'entête X-Network= CSD aux requêtes asynchrones simples.

Nous avons également calculé la taille des chaines de caractère avant et après compression.

Les résultats des nos essais sont affichés dans le tableau ci-dessus, nous constatons que pour les petites requêtes les temps donnent des résultats très flucutant. En revanche , des requêtes de 50'000 caractères. on remarque que le temps des requêtes compressées est plus court que celui des requêtes compressées. Enfin, nous constatons que du côté des requêtes simples, on se retourve face à un timeout dès 500'000 caractères, alors que pour les requêtes compressées on peut encore envoyer des requêtes à ce stade.

Ces résultats doivent être pris avec des pincettes, surtout pour les plus courtes requêtes, car les fluctuations du réseau influencent grandement les temps obtenus.

En ce qui concerne les longueur des requêtes envoyées au serveur, nous constatons que le gain est de plus en plus élevé avec la taille de la requête qui augmente. Pour une requête d'un seul caractère, il est inutilse et même contre productif de compresser la requête, car l'algorithme de deflate stocke des informations supplémentaires pour pouvoir par la suite décompresser la chaîne de caractères. En revanche, on observe une diminution de la taille des données à envoyer à partir des chaines de 500 caractères et on remarque que pour 500'000 caractères, la requête compressée ne fait plus que 15% de sa taille de base.