Systèmes mobiles – Laboratoire no. 2

Lara Chauffoureaux, Tano Iannetta, Wojciech Myszkorowski

17 novembre 2017

Tout le code source de l'application est joint à ce rapport. Celui-ci est également disponible sur GitHub via le lien suivant :

https://github.com/CoolPolishGuy/SYM-Protocoles-mobiles

Question 4.1

Actuellement notre application ne fait rien si le serveur est injoignable ou s'il retourne une erreur quelconque. Seul un message d'erreur est affiché à l'utilisateur, par exemple :

- *Not found* si le serveur renvoie une erreur 404.
- *Server internal error* si le serveur rencontre une erreur (à la décompression par exemple).

Aucune retransmission n'est effectuée et c'est un problème dans le cas d'une application en production. Dans ce cas, il faut que le client n'aie pas à gérer les erreurs du serveur. Si une erreur survient la requête devrait être stockée et renvoyée à un moment plus opportun pour le serveur.

Plusieurs solutions à ce problème sont envisageables et pour chacune d'entre-elles cette première étape est nécessaire :

Pour chaque requête ayant rencontré une erreur, stockage de celle-ci dans une liste

1. Essayer de renvoyer les requêtes après un certain temps fixe.
2. Si une requête passe sans erreur, en profiter pour envoyer toutes celles stockées dans la liste.
3. Dans le cas où le serveur nous enverrai des informations de temps en temps, attendre que celui-ci nous contacte pour tenter de vider la liste.
4. Si c'est un problème d'accès au réseau, il est possible de détecter quand le téléphone se connecte au réseau pour réessayer.

Une fois les requêtes transmises les étapes suivantes sont nécessaires :

- Sortir de la liste les requêtes qui ont pu être envoyées
- Notifier l'utilisateur, par exemple à l'aide des notifications Android

Un exemple de code de ce qui pourrait être fait se trouve sur la page suivante. Nous avons choisi de montrer la deuxième solution et de ne pas implémenter ce code directement dans notre application car cela rajoute quand même une certaine complexité.

La nouvelle classe *RequestUtils* ressemblerait à ça (tout le code n'est pas fourni, les changements et ajouts sont en orange) :

```
public class RequestUtils {

  private static String[][] pendingRequests = new String[][3]; // Array containing pending requests
  private static counter = 0;

  public static String sendRequest(String request, String url, String contentType) {

    (...)

    // Response code recuperation
    int responseCode = connection.getResponseCode();

    // If everything went well
    if (responseCode == HttpURLConnection.HTTP_OK) {
      // Reading response
      BufferedReader in = new BufferedReader(new
          InputStreamReader(connection.getInputStream()));
      String line;

      while ((line = in.readLine()) != null) {
        response += line;
      }
      in.close();

      // On retente le lancement des differentes requetes
      sendPendingRequests();
    }
    else {
      // We add the request in the pending array
      pendingRequests[counter][0] = request;
      pendingRequests[counter][1] = url;
      pendingRequests[counter][2] = contentType;
      counter ++;
    }

    (...)
  }

  // New function used to send the pending requests
  private void sendPendingRequests() {
    for(int i = 0; i < counter; i++) {
      // Same manipulation as belove to send the request
      (...)

      // Response code recuperation
      int responseCode = connection.getResponseCode();

      // If everything went well
      if (responseCode == HttpURLConnection.HTTP\_OK) {
        pendingRequests.remove[counter];
        counter --;
        // Notify the user with android notification
      }
    }
  }
}
```

Question 4.2

Il est possible d'utiliser un protocole asynchrone pour l'authentification d'un utilisateur. Toutefois, imaginons que nous utilisons une authentification à deux facteurs et que la réponse du serveur tarde trop à être reçue, il est possible que le 2ème facteur d'authentification échoue à cause de ce délai.

Il faut également que les transmissions se fasse dans le bon ordre. Les tâches asynchrones ne s'exécutent pas forcment dans l'ordre que l'on imagine. Il faut dans ce cas s'assurer du bon déroulement de l'authentification.

Il faudrait aussi s'assurer que les *credentials* restent les mêmes tout au long de l'authentification.

Pour une transmission différée les mêmes critères s'appliquent.

Question 4.3

On peut d'abord avoir une mauvaise synchronisation des threads entre eux. Surtout s'ils partagent des données mutuelles. Pour des raisons de programmation concurrente l'interaction des deux threads peut créer des problèmes de famine ou interblocage par exemple. En théorie, il est important d'avoir un thread par tâche, afin de bien diviser l'exécution des tâches qui se suivent.

Question 4.4

La solution multiplexée est beaucoup plus efficace. En effet, si on utilise une connexion pour chaque transmission nous aurons à chaque connexion, l'ouverture de celle-ci, l'envoie des données et puis la fermeture de la connexion.

En utilisant la solution multiplexée, nous allons ouvrir et fermer une seule connexion. L'inconvénient est que le serveur qui reçoit les données doit pouvoir gérer la réception de plusieurs données différentes.

Pour traiter la réponse du serveur nous pouvons imaginer d'avoir un id par "données" distinctes. De cette façon, le serveur peut utiliser l'id pour associer une réponse à un élément précis.

Question 4.5

a) Le JSON ne permet (pas encore) la validations de données. Par rapport au SOAP/XML cela a les impacts suivants :

Inconvénients

— La validation des données permet d'ajouter un couche de sécurité pour les données. Imaginons des données qui seraient saisies directement par un utilisateur (par exemple, le nom dans notre application), en JSON il serait facile de "injecter" des champs inattendus dans la sérialisation. En XML, avec une bonne DTD, un utilisateur malicieux ne pourrait pas ajouter des champs fictifs.

— Dans le cas où le but du développement est de fournir un service web pour des utilisateurs, le JSON devra forcément être accompagné d'une documentation précise afin d'éviter des problèmes. Malheureusement, souvent cela ne suffit pas et de nombreuses erreurs subsistent malgré cette documentation. Par contre avec XML, il est tout à fait possible de donner directement du WSDL pour générer automatiquement les données pour l'utilisateur et limiter donc les parties d'improvisation.

Avantages

— JSON est **beaucoup** plus léger et moins verbeux que le XML.
— Le JSON est très facile à utiliser et dans le cas d'une petite application où l'on contrôle les deux cotés de la communication, le JSON sera certainement plus aisé à mettre en place.
— JSON étant un sous-ensemble du Javascript, son utilisation avec le Javascript est facile et naturelle.

b) Oui, des protocoles tels que *Protocol Buffers* sont tout à fait utilisables en HTTP. Mais, l'utilisation d'un tel protocole dépendra des contraintes métiers. C'est comme avec le XML ou le JSON, cela dépend de ce qu'on a envie de réaliser à un assez haut niveau.

Néanmoins voici une petite liste des avantages / inconvénients par rapport à des protocoles de type JSON ou XML :

- Plus simple, plus léger que le XML malgré la possibilité d'une vérification presque aussi poussée.
- Sérialisation simple dans les langages supportés par rapport au XML.
- Quand même un peu plus verbeux que le JSON.
- N'est pas encore supporté avec tous les langages.
- *Protocol Buffer* ne dispose pas encore d'une intégration optimale avec les navigateurs (contrairement à JSON).
- "Grammaire" plus concise que le XML mais du coup, elle est aussi souvent plus compliquée à comprendre et à clarifier.

Question 4.6

Pour cette question , nous avons choisi de prendre le format (plain text) de différentes longueurs. Pour effectuer les calculs nous avons utilisé des tableaux de bytes dans lesquels nous avons calculé les longueurs de chaque string. Le 1er cas fut un texte de 703 bytes qui en mesurait 423 après la compression. Dans le 2ème cas, on passait de 1391 bytes à 818 et pour le 3ème, on passait de 4614 bytes à 2330.

La logique de la compression démontre une plus grande efficacité sur des textes plus grands. Les rapports constatés sont de 1.66 / 1.70 / 1.98 donc on peut effectivement constater que dans le 3 ème cas la compression est la plus efficace. Ces compressions ont été effectuées avec une compression de niveau 7 (selon la librairie).

Pour ce qui est de la partie théorique, il existe un procédé de plusieurs niveaux qui permet de calculer la compression. La documentation est accessible ici :

https://www.snellman.net/blog/archive/2015-06-05-updated-zlib-benchmarks/

On peut constater plusieurs choses. Tout d'abord, l'efficacité de compression va dépendre du type de fichier que l'on compresse.
En moyenne, la compression permet d'obtenir des tailles deux fois plus compactes et donc fait de la compression un outil indispensable pour une communication simple et efficace.