Systèmes mobiles

Laboratoire n°2 : Protocoles applicatifs

Auteurs : Loic Frueh - Koubaa Walid - Muaremi Dejvid
Enseignant : Fabien Dutoit
Assistants : Christophe Greppin, Valentin Minder
Date : 09.10.2018

1 Traitement des erreurs

Dans l'état actuel, ces interfaces ne savent pas traiter un message d'erreur. Par conséquent, plusieurs cas pathologique peuvent survenir.

Le cas d'une page d'erreur

Si le serveur est joignable mais qu'il retourne un message d'erreur quelconque, un message 4XX ou 5XX, ceci sera traité comme une réponse normale par notre application par conséquent nous afficherons le code source de la page html à la place du message attendu. Ceci peut être vu dans le cas où l'on envoie un xml qui n'est pas valide par rapport à la dtd ou à la place du résultat attendu on affiche un résultat incohérent.

Le cas ou le serveur n'est pas joignable

Le serveur peut ne pas être joignable pour différente raison, que ce soit un problème de temps de latence ou une erreur par rapport à son nom. Dans ces cas là, pour l'utilisateur il ne se passera rien du côté de l'interface, cependant, une exception sera levée par le code selon le type problème rencontré.

Une solution possible serait de rajouter un protocole qui va traiter toute les communications en entrée et sortie afin de valider la communication et afficher un résultat à l'utilisateur, selon les besoin.

2 Authentification

Si une authentification par le serveur est requise, peut-on utiliser un protocole asynchrone ?

On peut tout à fait mettre en place un système d'authentification en utilisant un protocole asynchrone. On peut obliger les clients à envoyer une token d'authentification à l'intérieur d'un JSON par exemple de chaque requête. Ainsi, le serveur pourra vérifier si le client est authentifié et si sa session est toujours valide.

Bien évidemment pour rendre ce type de méthode utilisable, il faudrait mettre en place un formulaire de connexion pour demander au serveur token d'authentification. Si le serveur valide la connexion du client, il lui renvoie un token d'authentification unique.

Quelles seraient les restrictions ?

Les restrictions serait de **mettre en place un formulaire de connexion** pour demander au serveur token d'authentification. Si le serveur valide la connexion du client, il lui renvoie un token d'authentification unique.

De plus il faudrait prévoir **un champ "session token"** que ce soit dans le xml ou json ou tout autre format pour garantir l'authenticité du client.

Peut-on utiliser une transmission différée ?

Le but de la transmission différée est d'envoyer une salve de message en une fois afin de libérer les ressources le plus longtemps possible. Dans le cas d'une authentification, qui est obligatoire à un moment précis, ceci ne serait pas du tout adapté. Bien que celle ci peut être utilisée, elle posera un problème de délai d'attente pour l'envoie et la réception de la réponse du serveur qui peut être très long avant de prouver l'authenticité du client qui serait très handicapant pour l'utilisateur.

3 Threads concurrents

L'exécution d'un thread A avant un thread B dépend de la rapidité d'exécution et traitement de chaque étape et il se peut qu'une étape s'exécute plus vite que les autres. Le traitement des données ne se fait donc pas dans l'ordre espéré et peut poser problème.

Il faut faire attention aux sections critiques et aux variables partagées, et s'assurer que l'ordre d'exécution de chacune des étapes (préparation, envoi, réception) soit bien respecté.

Par exemple, pour l'authentification évoquée plus haut, il faudrait éviter que le client n'envoie une requête avant d'avoir reçu la clé de session et vérifié l'authenticité de l'envoyeur.

4 Écriture différée

Effectuer une connexion par transmission différée

En effectuant une connexion par transmission on est sûr d'obtenir une réponse du serveur avant la suivante, ceci peut être utile dans le cas ou on utilise une connexion qui n'est pas stable et que l'on risque d'envoyer des paquets bien trop lourd d'une autre manière. Cependant, ceci peut également poser des problème dans le cas ou le réseau est stable et que l'on modifie plusieurs fois la même cellule étant donné que l'on ne peut garantir l'ordre d'arrivée de nos requêtes et l'état futur de des données peut être compromis sans utiliser un timestamp de modification.

Multiplexer toutes les connexions vers un même serveur en une seule connexion de transport.

Lors d'un multiplexage, on ouvre un canal entre le client et le serveur que l'on va utiliser pour communiquer entre eux. Chacun peut profiter de la réponse de l'autre pour continuer la conversation. De cette manière, on se rapproche d'une connexion active, ceci peut être embêtant dans le sens ou nous allons beaucoup plus consommer l'autonomie du téléphone mais au moins on garantit une meilleure fiabilité des données. Il faut également faire attention à la taille des paquets qui peuvent être problématique si ils sont trop volumineux par rapport à l'état du réseaux.

5 Transmission d'objets

REST/JSON sans validation vs SOAP

Les deux méthodes ont des avantages et des inconvénient. Dans le cas d'une infrastructure REST, l'évolution de l'application n'est pas limitée par une DTD ou autre validation quelconque, aussi, si la totalité des données n'est pas essentiel dans notre projet et que l'on a besoin d'énormément de données brutes par exemple un ballon météo qui n'a pas une connexion fiable mais envoie énormément de données une structure JSON non validée serait suffisante. Cependant, dans le cas ou l'on attend des données peu nombreuses mais très précise

ceci sera problématique car il faudra les valider manuellement afin d'éviter de faire crasher l'application par exemple, et cette vérification manuelle a un coût bien plus élevé que de profiter d'une DTD ou autre que le fera pour nous.

Protocol buffer et JSON ou XML

Bien sûr, protocol buffer permet de sérialiser des données d'une manière particulière, rien ne nous empêche de l'utiliser avec le protocol HTTP qui au final va envoyer une série de bit à travers le réseau.

Protocol buffer

Avantages :

- Validation
- Rétro compatibilité
- Interopérabilité

Désavantages :

- Dense
- Illisible
- Indécodable sans la structure source

JSON

Avantages :

- Lisible
- Pas de structure fixe
- Interopérabilité
- Peu verbeux

Désavantages :

- Pas toujours validable

XML

Avantages :

- Lisible
- Structure définie dans une dtd
- Interopérabilité
- Validation simple et rapide

Désavantages :

- Très verbeux

Optimisation de GraphQL

Une idéé serait de pre-enregistrer certaine requete directement dans le client, et ce pour gagner du temps et un bon moyen de combattre les fluctuation capricieuse de la connexion mobile. Aussi d'autres ameliorations possible serait de :

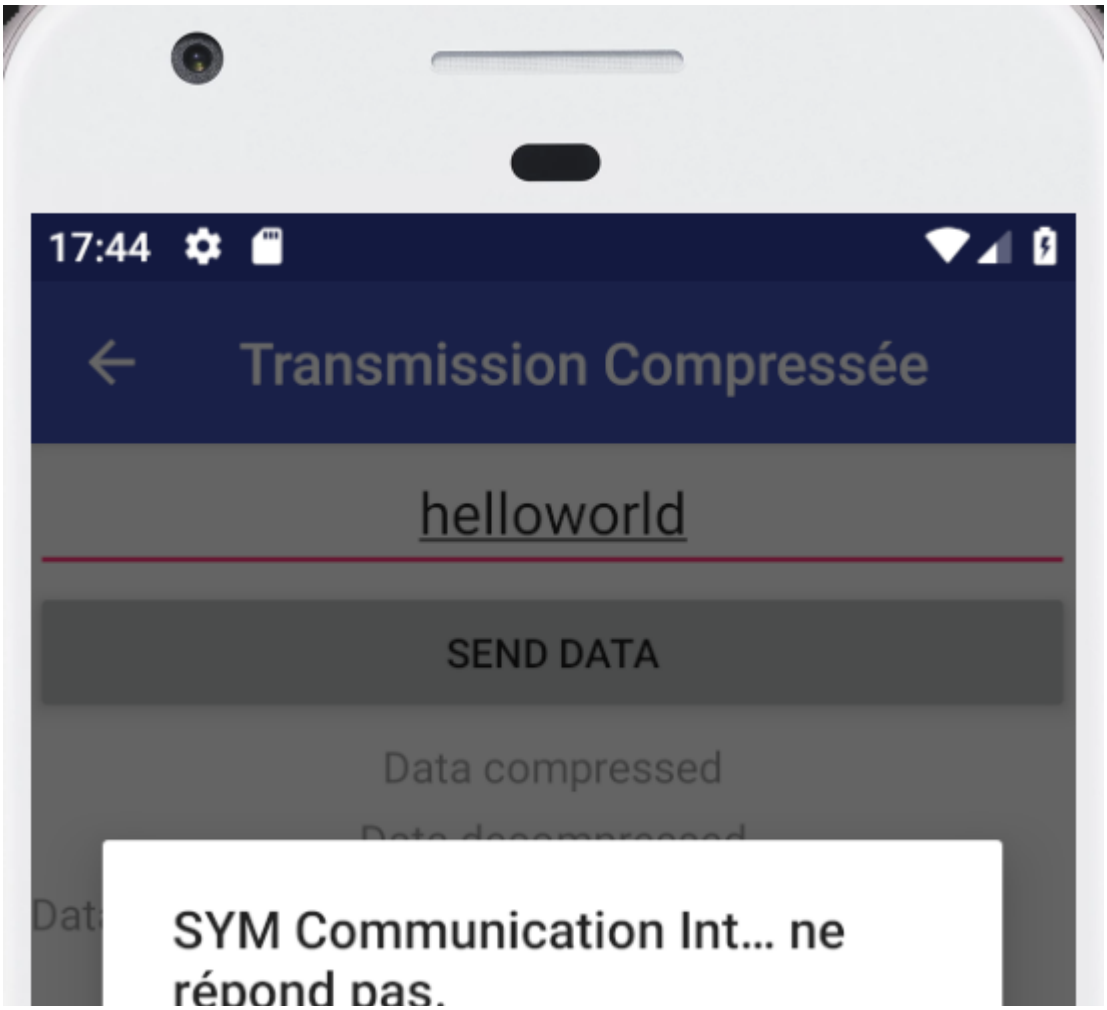
- Limiter le nombre de data
- Valider les data
- Optimiser les requêtes
- queries server-server au lieu de client-server
- utiliser des dataloader pour minimiser les accès db
- Si on touche la même table de la db plusieurs fois, on mets les appels en lots
- Si on touche plusieurs fois au même objet dans la db, on utilise celui qui est en cache

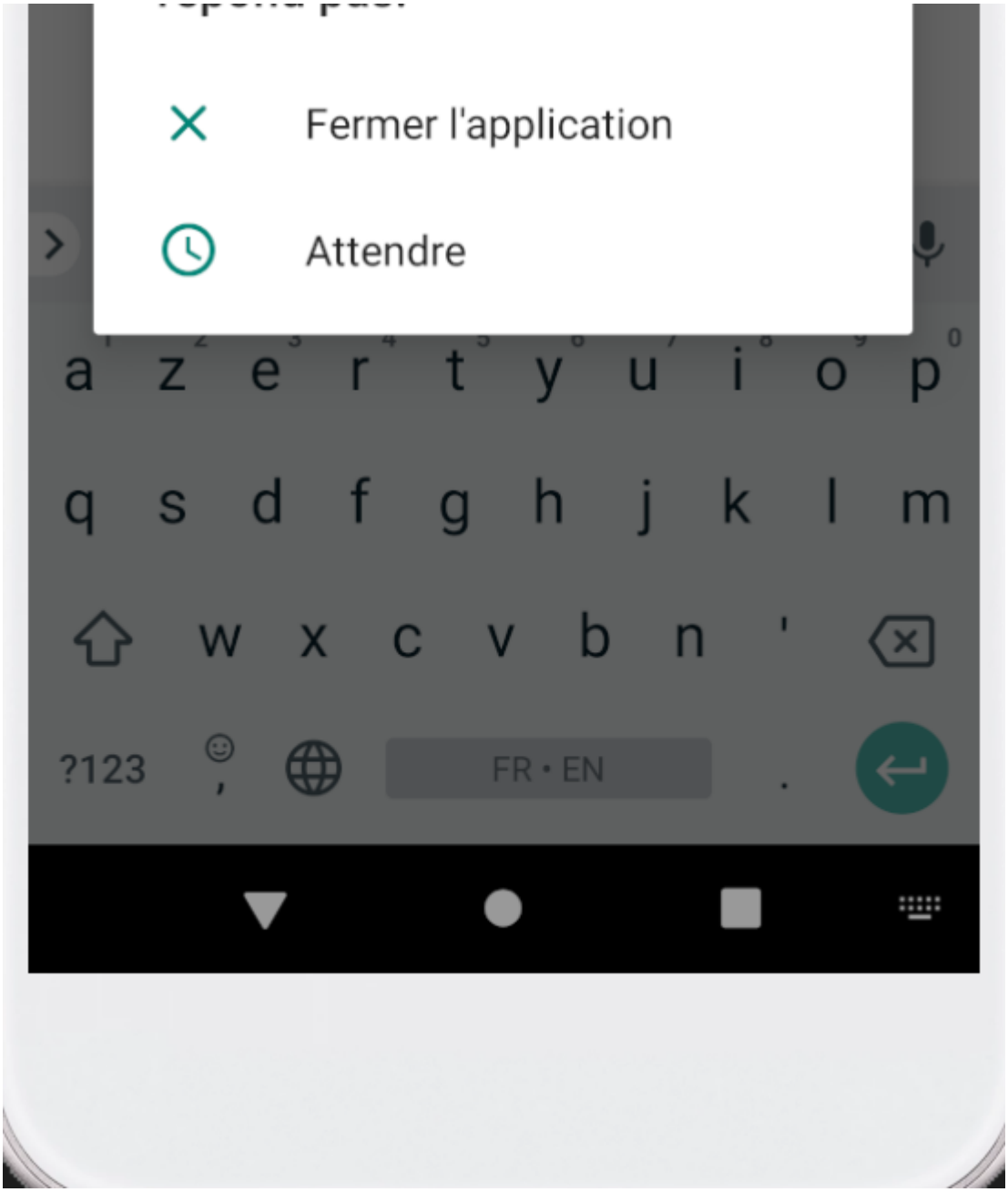
exemple d amélioration pratique possible : https://blog.apollographql.com/improve-graphql-performance-with-automatic-persisted-queries-c31d27b8e6ea

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 ?

L'activity de compression bien que normalement bien implementée n'envoie pas de requete au serveur faute d'une erreur de connexion internet non trouvée (alors que celle ci fait appel à une requete POST asynchrone qui fonctionne au meme moment). Le code fait bien appel à deflate et inflate pour compresser et decompresser. (nous avons tout essayer pour faire fonctionner la requete transmise au serveur mais sans succès, merci de bien vouloir regarder notre code source de l'activity en question, preuve de notre bonne foi)





Neanmoins nous avons pu tester le rapport de compression entre les données textes avant compression et après compression en local en utilisant la compression DEFLATE (sans requete au serveur).

- Pour du texte brut, nous avons par exemple pour un texte très court "Hello" le rapport de compression est proche de 1.5 tandis que pour un texte assez long (300 à 400 caractères) le ratio est de 2.5.
- Pour du json, le gain est plus élevé car les accolades sont très present pour chaque element. Pour un json simple d'une trentaine de caractères, le ratio est de 2.9
- Pour de l' xml, le gain est aussi élevé car les chevrons des balises sont assez recurrents. Avec un xml simple d'une cinquantaine de caractères, le ratio était de 2.7

Vous comparerez vos résultats par rapport au gain théorique d'une compression DEFLATE, vous enverrez aussi plusieurs tailles de contenu pour comparer.

En théorie une compression DEFLATE possède un ratio de 2.9 à 3.3 (c'est a dire que pour un texte de 120 caractères, le texte compressé contiendra 40 à 30 caractères environ).

Bien évidemment la pratique est assez éloigné de la théorie en fonction de la taille du texte à compresser. Par exemple pour un texte très court "Hello" le rapport de compression est proche de 1.5 tandis que pour un texte assez long (300 à 400 caractères) le ratio est de 2.5.

Comme spécifié plus haut cela varie de caractères repeétes dans le texte à compresser, en effet les **balise xml** et les **accolades pour le json** font que le **ratio de compression est en general très elevé**.

Le taux de compression sur un texte (txt/xml/json) varie donc en moyenne entre 60% et 80%.