# Labo 1 – Protocoles applicatifs mobiles

## Manipulation 1

***Quel gain, en volume et en temps, peut-on constater en moyenne sur les données échangées (xml, json et aussi protobuf) en utilisant la compression mise en place au point 1.4 ?***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nombre de données** | **JSON** | | | | | | | |
| **NONE** | | | | **DEFLATE** | | | |
| **Received** | **Payload** | **Temps 2G** | **Temps 5G** | **Received** | **Payload** | **Temps 2G** | **Temps 5G** |
| 3 | 460 | 460 | 300 | 100 | 200 | 460 | 150 | 90 |
| 10 | 2000 | 2000 | 500 | 250 | 400 | 2000 | 380 | 200 |
|  |  |  |  |  |  |  |  |  |
| **Nombre de données** | **XML** | | | | | | | |
| **NONE** | | | | **DEFLATE** | | | |
| **Received** | **Payload** | **Temps 2G** | **Temps 5G** | **Received** | **Payload** | **Temps 2G** | **Temps 5G** |
| 3 | 500 | 500 | 350 | 130 | 260 | 500 | 160 | 100 |
| 10 | 1800 | 1800 | 650 | 130 | 500 | 1800 | 200 | 130 |
|  |  |  |  |  |  |  |  |  |
| **Nombre de données** | **Protobuf** | | | | | | | |
| **NONE** | | | | **DEFLATE** | | | |
| **Received** | **Payload** | **Temps 2G** | **Temps 5G** | **Received** | **Payload** | **Temps 2G** | **Temps 5G** |
| 3 | 90 | 90 | 110 | 100 | 80 | 90 | 110 | 90 |
| 10 | 400 | 400 | 240 | 110 | 300 | 400 | 180 | 100 |

On peut constater que quand les données sont compressées, à la réception il y bien une réduction notable de la taille des données reçues :

* JSON :
  + Avec 3 données une réduction d’environ 50%
  + Avec 10 données une réduction d’environ 80%
* XML :
  + Avec 3 données une réduction d’environ 50%
  + Avec 10 données une réduction d’environ 75%
* Protobuf :
  + Avec 3 données une réduction d’environ 10%
  + Avec 10 données une réduction d’environ 25%

Concernant les temps d’exécution entre 2G et 5G nous pouvons constater que les différences sont surtout notables en 2G qu’en 5G en JSON et XML (environ 50%) alors qu’en 5G les gains de temps ne sont pas très notables. Les gains de temps obtenus avec le protobuf sont aussi plus faibles.

***Est-ce utile de compresser dans tous les cas ?***

Non, ce n’est pas utile de le faire dans les tous les cas. On peut notamment remarquer dans le cas du protobuf où les gains de temps ou de tailles sont très minimes (de 90 à 80 bytes et pratiquement le même temps pour 3 éléments). On peut donc se rendre compte que la compression serait plus judicieuse d’utiliser dans des grands sets de données surtout en JSON et XML. C’est dans ces deux formats de données que nous avons remarqué les plus grandes réductions. Cependant, dans notre cas, même avec des réductions de 50% ou plus les tailles sont minuscules (gain de 260 bytes et 60 ms pour le JSON en 3 éléments) c’est négligeable comme gain absolu.

## 2.1 Question théorique

L’api graphql mise à disposition aurait besoin d’une fonction qui retourne les livres d’un author selon son ID. Malheureusement elle n’est pas disponible et donc nous devons utilisé d’autre fonction pour avoir ce résultat.

J’ai aussi remarqué que le type de données pour les dates était un String ce qui n’est pas optimale mais fonctionne bien dans ce cas de figure.

Il serait intéressant que la pagination soit intégrée au requêtes.

Les fonctions countAuthors et countBooks ne sont pas indispensable mais sont un petit +.

Autrement j’ai bien aimé la simplicité des requêtes.

## Manipulation 3

***Veuillez expliquer à quoi sert le token obtenu avec la méthode onNewToken du Service Firebase de réception des messages.***

La méthode est appelée lorsqu’un nouveau token pour le projet par défaut est généré. Cette méthode est invoquée après l’installation de l’application lorsqu’un token est généré pour la première fois, et à nouveau si ce dernier change. Le token obtenu avec la méthode onNewToken est utilisé pour plusieurs raisons importantes :

* **Identifier l’appareil** : Chaque instance de votre application sur un appareil a un token unique, ce qui permet à Firebase de délivrer des messages à cet appareil précisément.
* **Gérer les mises à jour du token** : Le token peut être mis à jour pour diverses raisons, par exemple si la sécurité du token précédent a été compromise. La méthode onNewToken est appelée dans ces cas pour vous permettre de gérer ces mises à jour.
* **Prévenir la perte de messages** : Si vous vérifiez le token actuel uniquement après le lancement de l’application, vous risquez de perdre l’événement de mise à jour du token lorsque votre application est en arrière-plan1. Vous ne pourrez pas recevoir de messages push distants de votre serveur jusqu’à ce que l’utilisateur relance l’application et que vous envoyiez le nouveau token au serveur. La méthode onNewToken peut également être appelée pendant que votre application est en arrière-plan pour éviter de perdre des messages.

***Quand est-ce que ce token est généré ou regénéré, et qu’est-ce qu’une application doit faire avec celui-ci, vous illustrerez votre réponse en prenant l’exemple d’un service de messagerie, tel que WhatsApp.***

Lors du premier démarrage de l’application, un nouveau token est généré. Chaque fois qu’un token existant change. Cela peut se produire dans les scénarios suivants :

* L’application est restaurée sur un nouvel appareil.
* L’utilisateur désinstalle/réinstalle l’application.
* L’utilisateur efface les données de l’application.

Certaines défaillances matérielles de l’appareil peuvent entraîner la création de nouveaux tokens. Firebase peut également forcer le changement des tokens en appelant cette méthode dans le cadre de ses opérations.

Dans le contexte d’une application de messagerie comme WhatsApp, le token est utilisé pour envoyer des notifications push à l’utilisateur sur l’appareil spécifique où l’application est installée. Par exemple, lorsqu’un message est reçu, une notification push est envoyée à l’instance de l’application sur l’appareil de l’utilisateur via le token Firebase.

***Si l’utilisateur dispose de son application de messagerie sur plusieurs appareils, par exemple sur son smartphone et sa tablette, comment doit-on gérer les tokens obtenus sur chaque device ?***

Il est important de gérer ces tokens correctement pour assurer une livraison précise des messages. Voici comment vous pouvez gérer les tokens obtenus sur chaque appareil :

* Récupérez les tokens d’enregistrement de FCM et stockez-les sur votre serveur. Un rôle important du serveur est de garder une trace du token de chaque client et de maintenir une liste à jour des tokens actifs.
* Maintenez la fraîcheur des tokens et supprimez les tokens obsolètes. En plus de supprimer les tokens que FCM ne considère plus comme valides, vous pouvez vouloir surveiller d’autres signes que les tokens sont devenus obsolètes et les supprimer de manière proactive.
* Sur le démarrage initial de votre application, le SDK FCM génère un token d’enregistrement pour l’instance de l’application client. Ce token doit être inclus dans les demandes d’envoi ciblées à partir de l’API, ou ajouté aux abonnements aux sujets pour cibler les sujets.