Boutreaux Julien

Bertrand Hugo

Raport RT0805 golang :

I-Client.go :

Description générale :

Le client communique avec le serveur en utilisant grpc sur le port 8089. Le client est celui qui se charge de lire et de traiter les données json des fichiers donne_journee.json contenu dans le dossier données. Le client lit les fichiers un par un, pour chaque fichier, le client lis et extrait les données et créee une requête grpc.

Structure utilisée :

Le client utilise deux structures importantes pour extraire les données des fichiers json.

Les données dans les fichiers json sont organisés comme ceci :


```
[{"device_name":"c1153f7a-b060-4215-bf22-601e8f8e704c","operations":
[{"type":"DELETE","has_succeeded":false},{"type":"CREATE",
"has_succeeded":true},{"type":"DELETE","has_succeeded":true},
{"type":"CREATE","has_succeeded":true},{"type":"DELETE",
"has_succeeded":true},{"type":"UPDATE","has_succeeded":false},
{"type":"CREATE","has_succeeded":true},{"type":"UPDATE",
"has_succeeded":true},{"type":"DELETE","has_succeeded":true},
{"type":"UPDATE","has_succeeded":true},{"type":"DELETE",
"has_succeeded":false},{"type":"UPDATE","has_succeeded":true},
{"type":"CREATE","has_succeeded":true},{"type":"UPDATE",
"has_succeeded":true},{"type":"CREATE","has_succeeded":true}]},
```


Pour une machine, on a plusieurs opérations qui ont réussit ou échoués. Nous avons donc choisi de créer deux structures pour récupérer les données.

La structure opération :

Elle contient un champ Type (String) et un champ Has_succeeded (booléen)

Par exemple, une opération CREATE qui a réussi donnera :

Type = « CREATE »

Has_succeeded = True

La structure Data :

Elle contient un champ Device_name (String) et un champ Opérations qui est une liste d'objet Opération ([]Operation).

Avec cette structure on peut contenir les différentes opérations qu'une machine a effectués.

Fonction LireFichier :

Elle prend en paramètre le nom du fichier et renvoie une liste de la structure Data.

Elle commence par ouvrir le fichier, puis va ensuite utiliser un décodeur :

```
if err := json.NewDecoder(file).Decode(&data); err != nil {
    log.Fatal(err)
}
return data, nil
```

Il décode le contenu json dans la variable data, et si une erreur se produit lors du décodage alors cela veut dire que ce n'est pas le bon format de donnée et la fonction renvoit une erreur.

Fonction Extract_data :

Cette fonction s'occupe du traitement des données récupérées, en effet au lieu d'envoyer les données en brut au serveur, le client allège et traite les données. Cette fonction prépare la requête grpc.

Pour chaque machine, il y a 3 opérations possibles, CREATE, DELETE et UPDATE :

```
operation_create := &pb.Operation{}
operation_delete := &pb.Operation{}
operation_update := &pb.Operation{}

operation_create.Type = "CREATE"
operation_delete.Type = "DELETE"
operation_update.Type = "UPDATE"
```

La fonction va ensuite parcourir la variable data (la variable renvoyer par LireFichier) et va ensuite compter pour chaque opération, les réussites et les échecs.

```
for _, op := range d.Operations {

    if op.Type == "CREATE" {
        if op.Has_succeeded {
            create_has_succeed++
        } else {
            create_has_not_succeed++
        }
    }
    if op.Type == "DELETE" {
        if op.Has_succeeded {
            delete_has_succeed++
        } else {
            delete_has_not_succeed++
        }
    }
    if op.Type == "UPDATE" {
        if op.Has_succeeded {
            update_has_succeed++
        } else {
            update_has_not_succeed++
        }
    }
}
```

Le nombre de réussite et d'échecs pour chaque type d'opérations sera ajouter à la requête grpc. Le stockage sera ainsi plus lisible et pratique à lire.

Fonction RunClient :

Elle sera appelé dans le main.go, elle démarre le client se connecte au serveur sur le port 8089.

Ensuite elle va lire un par un les fichiers journée.json avec une boucle for, elle va les lire avec la fonction LireFichier, traiter les données avec la fonction Extract_data et envoyer la requête grpc au serveur. Le client s'arrêtera quand il aura fini de traiter tous les fichiers json.

Server.go :

Description générale :

Le serveur va écouter sur le port 8089, et quand il recevra une requête de la part du client, il traitera cette requête et stockera les données de cette requête dans une base de données mongoDB.

Fonction Create :

C'est la fonction qui est appelé lorsque le serveur reçoit une requête du client, il va extraire le jour et va appeler la fonction StroreToMongoDB pour chaque machine contenue dans la requête.

Fonction StoreToMongoDB :

Elle prend en paramètre la variable data qui est une structure *pb.Device et le jour. Elle se connecte au mongoDB avec l'identifiant root et le mot de passe root sur le port 27017 et va ensuite concaténer donnee_journee et le jour qu'on a spécifié en paramètre. De cette manière on peut créer (si elle n'est pas créée) ou choisir la bonne collection dans la base de données « Donnee_Projet ». Et elle insert les données dans la collection choisit

File_transfer.proto :

Description générale :

Ce fichier permet la mise en place des différentes structures pour l'envoi des données du client vers le serveur.

```
message Operation {
    string type = 1;
    int32 has_succeed = 2;
    int32 has_not_succeed = 3;
}
message Device {
    string device_name = 1;
    repeated Operation operations = 2;
}

message Request {
    int32 day = 1;
    repeated Device devices = 2;
}

message Response {}

service File_transfer {
    rpc Create (Request) returns (Response);
}
```

La structure opération ici compte pour une opération le nombre de réussite et le nombre d'échecs, un device contient une liste d'opération mais cette fois seulement 3 opérations (CREATE, UPDATE et DELETE) qui ont respectivement chacune un nombre de réussites et d'échecs. Chaque requête faite au serveur sera constituée du numéro du jour ainsi que d'une liste de device dont la structure vient d'être présentée. On a choisi de faire une requête par jour et non par device pour limiter le nombre de requêtes faites au serveur.

Cette approche permet de simplifier et de réduire au maximum les données ainsi que de les rendre beaucoup plus lisibles avant de les envoyer au serveur, et de les envoyer en une fois (une par jour) limitant le nombre et le poids des communications. Ainsi le serveur pourra directement mettre dans sa base de données les données reçues sans plus de traitement.

Résultat MongoDB :

Pour voir les résultat sur mongoDB, on se connecte en ligne de commande :

mongo --username root --password root --authenticationDatabase admin

on peut faire show dbs pour voir les différente base de donnée :

```
> show dbs
Donnee_Projet   0.000GB
admin           0.000GB
config          0.000GB
local           0.000GB
```

ensuite on fait : use Donnee_Projet pour aller dans la base de donnée

Puis on fait show collections :

```
> show collections
donnee_journee_1
donnee_journee_2
donnee_journee_3
donnee_journee_4
donnee_journee_5
```

Et ensuite pour vérifier le contenu on fait db.donnee_journee_1.find() :

donnée mongoDB :


```
> db.donnee_journee_1.find()
{ "_id" : ObjectId("663b949ba2428425e264478d"), "device_name" : "c1153f7a-b060-42
15-bf22-601e8f8e704c", "operations" : [ { "type" : "CREATE", "has_succeed" : 5, "
has_not_succeed" : 0 }, { "type" : "DELETE", "has_succeed" : 3, "has_not_succeed"
 : 2 }, { "type" : "UPDATE", "has_succeed" : 4, "has_not_succeed" : 1 } ] }
{ "_id" : ObjectId("663b949ca2428425e264478f"), "device_name" : "971645e6-6870-4d
b6-9e6b-817227d8f338", "operations" : [ { "type" : "CREATE", "has_succeed" : 11,
"has_not_succeed" : 3 }, { "type" : "DELETE", "has_succeed" : 5, "has_not_succeed
" : 2 }, { "type" : "UPDATE", "has_succeed" : 8, "has_not_succeed" : 1 } ] }
```


Donnée journée_1.json :


```
Projet_RT0805 > donnees > {} journee_1.json > {} 2 > [ ] operations > {} 0 > has_succeeded
1   [{"device_name":"c1153f7a-b060-4215-bf22-601e8f8e704c","operations":
    [{"type":"DELETE","has_succeeded":false},{"type":"CREATE",
    "has_succeeded":true},{"type":"DELETE","has_succeeded":true},
    {"type":"CREATE","has_succeeded":true},{"type":"DELETE",
    "has_succeeded":true},{"type":"UPDATE","has_succeeded":false},
    {"type":"CREATE","has_succeeded":true},{"type":"UPDATE",
    "has_succeeded":true},{"type":"DELETE","has_succeeded":true},
    {"type":"UPDATE","has_succeeded":true},{"type":"DELETE",
    "has_succeeded":false},{"type":"UPDATE","has_succeeded":true},
    {"type":"CREATE","has_succeeded":true},{"type":"UPDATE",
    "has_succeeded":true},{"type":"CREATE","has_succeeded":true}]},
```


On voit bien que pour la machine : « c1153f7a-b060-4215-bf22-601e8f8e704c », il y a 5 opérations CREATE et elles ont toute réussies.

Main_test.go :

Pour la partie test, nous avons estimé que les seules fonctions pertinentes à tester sont lire_fichier et extract_data qui sont dans client.go. Tester ces deux fonctions permet de couvrir le code de client.go à 60%

Pour tester ces fonctions, nous avons mis en place un fichier de test test.json qui contient des données similaires aux journées.

On teste d'abord lire_fichier puis si le retour est bon on peut tester extract data qui se base sur le retour de lire fichier et on vérifie avec les valeurs attendues, en comparant une par une les différentes valeurs contenues dans les structures.

Ceci conlu notre rapport.