Summit en Web-App moderne

Installation :

L'installation globale du projet se fait grâce à VCPKG. VCPKG requiert PowerShell pour fonctionner. L'installation de **VCPKG** et de **PowerShell** devrait être possible sans l'aide du support informatique. Pour ce faire, il faut **modifier** les paramètres du **proxy** :

http://proxy-wcg.csf.sfil.fr:8080

VCPKG va nous permettre de télécharger **protobuf** et **gRPC**. On utilisera dans PowerShell la commande (pensez à mettre vcpkg dans le **path**) :

vcpkg install … [grpc/protobuf protobuf :x64-windows]

Le tout devrait être installé à l'endroit suivant :

../localapp/apps/vcpkg/vcpkg/packages

Pour que le tout fonctionne, il faut correctement setup le **path** et les **paramètres du projet** afin que toutes les bibliothèques externes soient importées au sein du projet.

Pour GraphQL et gRPC en Python, on utilisera **pip**.

Après avoir installé et configuré pip, on exécutera les commandes suivantes :

pip install grpcio

pip install grpcio-tools

pip install flask ariadne flask-sqlaclhemy flask-cors

Ainsi, nous pourrons utiliser **gRPC**, **Flask** et **GraphQL** (compris dans ariadne).

Enfin, j'ai utilisé openssl pour le cryptage des mots de passe. Il faut donc également l'installer. Vous pouvez utiliser VCPKG à nouveau.

Commandes utiles pour Protobuf

Génération des fichiers C++

protoc -I=*$path$* --cpp_out=*$path_out$ $path/file.proto$*

Génération des fichiers gRPC

protoc -I=*$path$* --grpc_out=*$path_out$* --plugin=protoc-gen-grpc=*$path/grpc_cpp_plugin.exe$ $path/file.proto$*

Génération des fichiers Python

python -m grpc_tools.protoc –I=*$path$* --python_out=*$path_out$* --pyi_out=*$path_out$* --grpc_python_out=*$path_out$* *$file.proto$*

Description du package

Le package comprend trois projets :

1. Sfil_export_ccy

La base C++ reliée à Summit. Ce projet est également le serveur gRPC et comprend des fichiers générés par protobuf. Ces fichiers sont reconnaissables car ils finissent par .pb.cc/.h. Nous avons également des fichiers gRPC reconnaissables par .grpc.pb.cc/.h. Ces fichiers sont également générés par protobuf.

a. Les fichiers

sfil_export_ccy

Fichier contenant le main.

ExportRouteImpl

Fichier de définition de la classe ExportRouteImpl qui représente notre serveur gRPC. On y trouve la fonction `ExportRouteImpl::GetCurrency` qui est le détail de la fonction présent dans le fichier *sfil_export_ccy_route.proto*. **Veuillez donc vous référer à cette dernière pour la structure des fonctions rpc de gRPC.** Ainsi, le message de réponse est stocké dans les paramètres de la fonction. Attention, le pointeur de la réponse est automatiquement affecté par gRPC.

Ce fichier définit également la fonction runServer qui est concrètement notre main. Cette fonction a pour objectif de setup les paramètres du serveur et de le lancer afin qu'il attende une requête de la part du client gRPC. On remarque ainsi que le serveur fonction sur le localhost:50051. Cette valeur est évidement modifiable selon vos besoins. Attention, il n'y pas de fonction d'arrêt il faut donc forcer l'arrêt en arrêtant l'exécution.

sfil_export_ccy.proto et summitType.proto

sfil_export_ccy.proto définit la structure de la donnée que l'on va exporter ici CCY. Se repose sur summitType.proto qui va définir tous les sous types de Summit utilisés dans la structure de CCY. L'objectif de ces fichiers est de reprendre la structure exacte de CCY définit dans son métamodèle au sein de Summit.

sfil_export_ccy_route.proto

Définit le service gRPC et les rpc qui va le composer. Plus simplement, ce fichier va définir les requêtes de notre serveur. Attention, un rpc ne peut avoir que des messages protobuf comme entrée et sortie.

b. L'exécution

Pour exécuter ce projet, il doit tout d'abord être ouvert au sein de Summit avec l'environnement PE_devenv. Enfin, il suffit d'écrire *sfil_export_ccy* au sein de la console d'environnement PE.exe pour que le projet s'exécute.

2. CurrencyGrpcClient

Ce projet est le client gRPC en Python. Ce projet ne contient pas de main et peut donc être fusionné avec le projet du serveur GraphQL qui va appeler la fonction `run(self,ccyname,username,password)`. L'autre fonction run permet de la tester directement depuis ce projet.

Cette fonction crée le channel de connexion avec le serveur et appelle la fonction rpc définie dans le fichier _route.proto du projet C++.

3. ExportCcyGraphQL

Ce projet est le serveur GraphQL utilisant Flask et donc Python.

a. Les fichiers

__init__.py

Définit l'application flask.

ExportCcyGraphQL.py

Ce fichier contient le main de notre projet. Il définit nos 2 requêtes web. La « Get » va afficher l'explorerGraphiQL. Ce dernier est un outil qui va nous permettre de simuler un client Web et donc de tester nos requêtes. La « Post » va effectuer les requêtes saisies dans l'explorerGraphiQL et présenter les données.

schemaCcy.graphql

Ce fichier définit la structure des données et les query possibles pour le GraphQL. Comme pour le protobuf, les types définis ici sont identiques à ceux présents dans le métamodèle au sein de Summit.

Queries.py

Ce fichier va définir les query indiqués dans le fichier .graphQL. Ainsi, il faut se référer à la fonction `@query.field("getCurrency")`
`def getPerson(_,info,ccyName,username,password):`

pour la rédaction des futures définitions de fonctions GraphQL. Cette fonction va appeler la classe CurrencyGrpcClient définit dans notre client gRPC afin d'exécuter le rpc lié à la requête et ainsi récupérer les données depuis Summit.

b. Exécution

Il suffit d'exécuter le projet, allait à l'adresse internet écrit et avoir le serveur gRPC qui tourne afin de pouvoir tester les query. Le nom d'utilisateur et le mot de passe de test sont test et test.

Futur pour ce projet :

1. Client web

La première étape pour finir ce projet est la réalisation d'un client web possiblement en React afin de remplacer l'explorerGraphiQL. Son objectif sera d'appeler les requêtes et d'afficher les résultats.

2. Généralisation pour Summit

Afin de généraliser la solution aux autres fonctionnalités de Summit, on pourrait modifier le genméta au sein de Summit afin qu'il génère à partir du métamodèle les fichiers protobuf et GraphQL. Ainsi, la généralisation serait semi-automatisée.