Manuel d'instructions

Bienvenue dans le manuel du projet de TP4 du cours d'Architecture logicielle et conception avancée.

Le manuel est divisé en quatres parties : l'API REST, le client, la base de donnée et le service Spark.

le serveur REST API

détail des choix d'architecture et de conception et des raisons derrières ceux-ci

le serveur est un REST API pure écris avec la technologie Node.

Node à été choisis plutôt que Java pour deux raisons principales. Premièrement, il s'agit d'une technologie pratique permettant de créer rapidement des serveurs HTTP grâce à sa librairie express et à son langage flexible. Entre autre, Node à l'avantage de nécessité moins de classe et de fichier que Java. De plus, avec Node, la gestion des requête HTTP suit un modèle asynchrone plutôt qu'une gestion de session par Thread, ce qui évite tous les problémes de concurence et bug du a des condition de course (race-condition). En second lieu, Node à été choisis car il s'agissait de la technologie étant la mieux connu des dévellopeurs du projet. Il est plus facile et fiable pour les dévellopeurs de travailler avec les technologies qu'ils connaissent le mieux.

Ce serveur suit la convention REST et le principe du HATEOAS (Hypermedia as the Engine of Application State). Produire un "true REST API" n'était pas un requis du projet. Nous concidérons que les principes REST, lorsque tous correctement respecter, apporte des avantages de maintenabilité et de fiabilité et de réutilisabilité. Il est possible avec ces principes de créer un client agnostique au modèles de donnée de l'API et de la base de donnée, ne réagissant uniquement au réponse des requêtes qui lui sont transmise à partir de la requête initial. Cela permet entre autre d'étendre et de modifier le modèle de l'API sans avoir a modifier le client. Le projet était une bonne occasion par sa taille et sa simplicité de mettre en pratique la théorie derriére les Api REST complet.

À ce sujet, un défi notable à été de gérer les requêtes demandant des calculs de Spark de manière purement REST. Pour se faire, le REST Api suit une stratégie de création de ressource temporaire. Faire une requête demandant de long calcul impliquera donc de faire un "POST" pour créer une requête de calcul. Ce message renverra un lien vers une ressource pour connaître l'état de la requête. Lorsque la requête a fini d'être traité, cette route envoie en plus la route ou les résultat des calculs peuvent être accédé. Le REST API garde pendant une durée limité le résultat des calculs fait a partir de tel requête. De plus, un nombre maximum de requête longue peut être exécuté en parallel. cette structure permet de garder les contraintes REST et d'exécuter des téches complexes et potentiellement longue sans bloquer le client et le serveur.

Ce projet de REST API n'est pas complet, en ce sens qu'il y a beaucoup d'améliorations qui pourrait y être faite, mais qui dépasserais largement le cadre initial du travail pratique. En particulier, l'api ne renvois pas suffisament de meta-donné sur la structure des donnée envoyé et attendu dans les requêtes. Il serait intéressant de rajouter dans les hypermedias des détails sur la forme du corps des requêtes attendus, de sorte que le client puisse traité ces informations et demander de manière plus précises les données à envoyé selon l'état de l'application. De plus, il aurait été intéressant de permettre l'envois et la réception de donnée de l'API sous plusieurs formes plutôt que seulement le JSON. Les données pourrait être envoyé en XML ou en HTML (pour être présenter visuellement directement). Aussi, cette api n'a aucune gestion de concurence entre les

clients. Il n'a pas de support pour la méthode PATCH. cette dernière fonctionnalité aurait été particulièrement utile, mais aurait nécessité un langage permettant de spécifier les modifications au donnée sous forme d'opération. Finalement, Le système de modification utilise la méthode "PUT", mais ne permet pas, contrairement à la spécification original de la méthodes HTTP, d'insérer des ressources à l'adresse indiquée si ces éléments n'ont pas été créer grâce à la méthode POST auparavant.

modèle de donnée et implémentation

Un des défauts actuel des API REST est l'absence de standardisations des requêtes et des réponses. Ainsi, une structure à dû être établie dans les requêtes et les réponses de l'API afin de permettre la création du client.

- Toutes les informations en requête et en réponses sont attendu et envoyé en JSON.
- Les réponses de l'API ont deux attributs principaux et commun : "result" contient les données directement lier à la requête, et "__hypermedia" contient l'information sur la liste de route pouvant être accédé suite à cette requête, permettant de modifier l'état de l'application.
- Le modèle des factures, le sujet de l'API, est envoyé et attendu sous la forme suivante : { id : number, produits : [{ nom : string, prix : number }] } lors d'une requête pour ajouté ou modifier une facture, c'est donc sous cette formes que doit être écris le JSON a envoyé

Mise en place du REST API.

1. Installer node en suivant les instructions sur le site https://nodejs.org/en/ la version 8 ou 10 est nécessaire.
2. Dans le dossier "api" du projet, ouvrir un terminal.
3. Dans le terminal, utiliser la commande "npm install" pour installer les dépendences
4. pour le fonctionnement du programme, la variable environnement "CASSANDRA_IP" doit contenir l'adresse ip de la BD cassandra (les détails de sa mise en place sont indiqué plus bas). 4.1 Il est possible de configurer les variables environnement sur linux en démarrant le programme de la sorte "CASSANDRA_IP=X.X.X.X node bin/www" 4.2 Dans un environnement window, il est possible d'écrire "set CASSANDRA_IP=X.X.X.X" avant de lancer le programme. 4.3 Alternativement, il est possible d'inscrire la valeur de la variable d'environnement dans le fichier ".env", qui sera lu au démarrage du programme. les valeurs assigner directement au variable d'environnement ont priorité sur celle écrite dans le fichier ".env"
5. Dans le terminal, utiliser la commande "node bin/www" pour démarer le serveur contenant le REST API

Le client.

choix d'architecture et de conception du clients

Le client a été un défi intéressant à concevoir pour ce projet. l'API suivant le principe HATEOAS, il n'est pas nécessaire pour le client d'enregistrer l'état de l'application ou de la logique propre au facture et à leur création. Le client est un utilitaire en ligne de commande intéractif permettant d'explorer n'importe quel REST API à condition que celui ci ne suivent la structure des requêtes et des réponses de l'API présenté plus haut.

voici quelque point à noté sur le client :

- Le client est agnostique au contexte du REST API et utilise uniquement les informations contenus dans les hypermedias pour permettre l'exploration de l'API et la présentation des prochains choix.

- Lorsque le client demande des données pour envoyé au serveur (dans le cadre d'une requête aillant la méthode POST ou PUT, généralement), il s'attend à des données sous forme de JSON uniquement.
- Il est un probléme connu que certain terminaux ont des problémes d'encodages empéchant l'écriture correct de certain charactéres accentués.

Mise en place du client.

1. Le client est un programme en ligne de commande écris en node. pour l'utiliser, il faut donc avoir Node 8 ou Node 10 installer. le site https://nodejs.org/en/ permet de téléchargé un installeur intéractif pour le système d'exploitation désiré.
2. Dans le dossier "client" du projet, ouvrir un terminal.
3. Dans le terminal, utiliser la commande "npm install" pour installer les dépendences.
4. Dans le terminal, utiliser la commande "node main.js --uri X.X.X.X:3000" pour démarrer le client. "X.X.X.X" représente l'adresse IPv4 du serveur contenant le REST API. Si ce dernier est sur la même machine que le client, ce sera donc localhost:3000.

À partir de là, un programme interactif en ligne de commande commencera, permettant d'intéragir avec le REST API.

Base de donnée cassandra

La base de donnée choisis est celle proposé par le projet initial : Cassandra. cette base de donnée utilise un langage nommé CQL pour faire des requêtes. L'équipe n'avais pas de connaissance préalable au projet sur cette base de donnée spécifique, mais celle-ci est suffisament simple et bien documenté pour pouvoir être mise en place sans difficulté excesive.

dans le cadre du projet, Cassandra ne contient qu'une seul table "factures", qui elle même a deux colonnes : id, un nombre (32 bit signé) utilisé comme clé primaire, et produits, une liste de tuple sous la forme (nom de l'article, prix).

ce modèle à été choisis pour sa simplicté et son implémentation rapide même en aillant une connaissance que superficielle de la base de donnée et de ses capacités.

Cela dit, il aurait pu être utile de créer une table de produit contenant le nom, l'id et le prix de tous les produits, et que la colonne produits de la table facture contiennemt plutôt des tuples sous la forme (id du produit, nombre de produit acheté de cette id). C'est le temps, l'échéancier et le manque de connaissance sur le sujet qui à empéché de restructurer la base de donnée et le modèle du serveur avec cette modifications.

Mise en place de la base de donnée

Créer une machine virtuelle avec une image ubuntu 16.04

Installation de Cassandra: Pour pouvoir installer Cassandra, il faut installer java. Vous pouvez le faire avec les étapes suivantes: 1- sudo add-apt-repository ppa:webupd8team/java 2- sudo apt-get update 3- sudo apt-get install oracle-java8-set-default 4- java -version (pas nécessaire, juste pour vérifier que vous avez installer java)

```
Par des problémes d'avertissement, il est nécessaire ajouter trois clés afin que
cassandra marche parfaitement:
        Pour commencer, vous pouvez chercher les repos suivantes afin d'avoir
```

```
accés aux clés
                1- echo "deb http://www.apache.org/dist/cassandra/debian 22x main"
| sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
                2- echo "deb-src http://www.apache.org/dist/cassandra/debian 22x
main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list

        Voici les commandes à executer pour installer les clés:
                1- gpg --keyserver pgp.mit.edu --recv-keys F758CE318D77295D
                2- gpg --export --armor F758CE318D77295D | sudo apt-key add -
                3- gpg --keyserver pgp.mit.edu --recv-keys 2B5C1B00
                4- gpg --export --armor 2B5C1B00 | sudo apt-key add -
                5- gpg --keyserver pgp.mit.edu --recv-keys 0353B12C
                6- gpg --export --armor 0353B12C | sudo apt-key add -

Après d'avoir les clés, updater les packages:
        1- sudo apt-get update

Enfin, installez Cassandra:
        1- sudo apt-get install cassandra
```

Configuration de Cassandra: Pour rouler un noeud cassandra pavec l'addresse ip de la machine virtuelle, suivez les étapes suivantes: 1- Aller sur le terminal et allez à /etc/cassandra 2- Modifier le fichier cassandra.yaml avec sudo (sudo vim cassandra.yaml) et changez les paramêtres suivants: *seed_provider: (...) parameters: -seeds: "127.0.0.1,"

```
                *listen_address: <addresse ip de la vm>

                *start_rpc: true
                et commentez les parametres rpc_address et broadcast_rpc_address
```

Ouvrir Cassandra Pour commencer Cassandra, executer cette commande sur le terminale: - sudo service cassandra start

```
Il est possible de verifier que le service est bien ouvert avec cette commande:
        - sudo nodetool status

Pour acceder au noeud, executer cette commande:
        - cqlsh <addresse-ip de la vm>

A partir de cette commande, vous pouvez faire toutes les requetes de cassandra
directement sur ce noeud. Vous pouvez
```

quittez en tout temps avec la commande "exit"

```
Pour fermer Cassandra, executer cette commande:
        - sudo service cassandra stop
```

Spark

l'outil Spark a causé beaucoup de difficulté à notre équipe de dévellopement, au point tel que l'outil n'est pas fonctionnelle dans la dernière version du produit.

Nous avons réussi a démarer le noeud "Master" ainsi que deux noeuds de traitement esclave (Slave). Cependant, beaucoup de temps ont été nécessaire pour permettre la communication des jobs jusqu'au noeuds Spark, sans résultat. Des recherches récente m'on indiquer que la seul méthode à suivre était de créer un script python pour envoyé à Spark expliquant comment mettre en place les configuarations pour envoyer des "Jobs". Le temps à manquer pour implémenter cette solution prométeuse pour faire fonctionner la communication.

Il est dommage que Spark n'est pas de REST API public permettant d'effectuer des "Jobs". Cela permettrait de ne pas être dépendant de quelque langage spécifique pour manipuler Spark mais plutôt de pouvoir utiliser n'importe quel outil capable de faire des requêtes HTTP.

manuel d'instruction

Note : Ce manuel d'instruction permet de créer un master et deux slave de traitement spark, pret a recevoir des "jobs" et a en retourné le résultat. Cependant, il n'indique pas comment configurer le système pour permettre d'envoyé des requêtes au master, puisque cette partie crutial n'a pas été résolu.

Installation et ajout de master à Spark 1. Aller sur le site spark.org 2. Télécharger spark-2.1.0-bin-hadoop2.4 3. Après avoir installé Spark, aller dans le dossier "conf" du répertoire Spark 4. Ouvrir un terminal et exécuter la commande "vi spark_env.sh" 5. Ajouter les 4 lignes de codes suivant: SPARK_MASTER_HOST=127.0.0.1 SPARK_LOCAL_IP=127.0.0.1 SPARK_WORKER_INSTANCES=2 SPARK_WORKER_CORES=1 6. Sauvegarder le fichier et quitter 7. Aller dans le dossier "sbin" 8. Ensuite, exécuter la commande "./start-master.sh" Nous pouvons alors trouver notre master sur le port 8080, donc on peut tout simplement y avoir accés en entrant "localhost:8080"

Ajout des workers 1. Copier le URL du master 2. Exécuter la ligne suivante "./start-slave.sh " et ensuite coller le URL master que vous avez copié (par exemple: ./start-slave.sh spark://127.0.0.1:7077) Vous pouvez maintenant rafraéchir la page web du master et vous pouvez maintenant voir que les 2 workers sont présents

...