Introduction à DataStore et CloudStore

Mise en place

Pour pouvoir utiliser DataStore et CloudStore, il faudra rajouter les bonnes dépendances[1] dans le fichier *pom.xml* de votre projet.
Vous pouvez soit exécuter votre application localement, soit la déployer dans le Cloud. **Par défaut**, même en déployant localement, DataStore utilisera le Cloud.

Configuration entièrement locale

Pour pouvoir travailler hors connexion, il faut

1. Installer un émulateur de DataStore[2].
2. Le configurer avec votre *project-id*
3. Démarrer l'émulateur
4. Récupérer les variables d'environnement et les mettre dans le terminal où vous lancez votre application

Configuration mixte (devsever et Store distant)

L'utilisation du DataStore nécessite que votre projet soit autorisé, ce qui se fait à travers la console de l'application

1. Allez dans l'*API Manager* de votre projet
2. Créez une *service account key* et téléchargez le fichier json généré. Attention à ne pas le perdre !
3. Dans le terminal où vous démarrez le dserver, mettez le chemin vers le fichier json dans la variable GOOGLE_APPLICATION_CREDENTIALS.

DataStore

Pour le reste de l'exercice nous vous conseillons l'utilisation de la documentation en ligne.

[1] https://cloud.google.com/appengine/docs/flexible/java/using-cloud-datastore
[2] https://cloud.google.com/datastore/docs/tools/datastore-emulator

Entités et requêtes simples

Tout objet dans le DataStore est représenté par une entité qui peut avoir une ou plusieurs propriétés. Une clé unique est associée à chaque entitée et peut être partiellement fixée par l'utilisateur.

1. Ajoutez à votre projet une DataStoreServlet qui répondra à un *GET* et un *POST* */datastore/*
2. Mettez en place le code permettant d'ouvrir une connexion au datastore (1 ligne).
3. Pour chaque *POST* votre servlet devra créer une nouvelle entité permettant de sauvegarder un Article (c.f TP1) avec les contraintes suivantes :
    a. Son *kind* sera "article"
    b. Les propriétés seront passées en JSON
    c. La clé sera générée aléatoirement (*KeyFactory*)
4. Pour chaque *GET*, votre servlet affichera toutes les entités de *kind* "article" actuellement dans le datastore.
5. Ajoutez une méthode permettant d'effacer une entité dont le nom aura été passé par POST.

Ancestor paths et transactions

DataStore fournit une notion d'arborescence à travers les *ancestor paths* qui permettent de lier hiérarchiquement des entités. Cela permet d'organiser les données de façon hiérarchique. Le reste du travail pourra être fait dans une servlet *DataStoreServletAncestor* afin de pouvoir comparer avec la précédente.

1. On veut maintenant que les articles soient associés à un magasin. Modifiez votre code pour refléter cette structuration lors de la création d'articles
2. Modifiez le GET pour ne retourner que les articles d'un magasin donné dont la quantité est inférieure à X. Le nom du magasin et X seront donnés en paramètre.
3. On souhaite augmenter de 10% les prix des articles d'un magasin donné. Proposez une méthode à l'aide d'une transaction.
4. Une limitation de la notion d'ancêtre est qu'elle est permanente. Proposez un moyen de faire passer un article d'un magasin à un autre.
5. Mesurez le temps d'écriture dans le DataStore pour les versions avec et sans ancêtre. Commentez le résultat.

CloudStore

Dans cet exercice nous allons étendre notre application de gestion de stock en associant des images à nos articles.

1. En utilisant l'interface web de CloudStore, regardez quelle est la qualité de service[3] du *Bucket* associé par défaut à votre application.
2. Ajoutez à votre servlet un mécanisme permettant de créer un fichier dans votre *Bucket.* Le contenu pourra être aléatoire.
3. Modifiez la création d'article pour y ajouter l'upload d'une image dans le CloudStore. L'image pourra être passée par un POST. Pensez à garder une référence à cette image quand vous sauvegardez l'article dans le DataStore
4. Modifiez le GET pour que soient retournés, avec les articles, l'URL associée à chaque image.

Pour les longues soirées d'hiver

1. La transformation d'un objet Java en une entité est relativement fastidieuse à faire à la main. Regardez donc du coté de l'API Objectify
2. Les requêtes sans ancêtres sont éventuellement cohérentes, contrairement à celles avec ancêtre qui sont strictes. Proposez un scénario permettant de le vérifier.

[3] https://cloud.google.com/storage/docs/storage-classes