TP : Redis

L'objectif de cet exercice est d'implémenter une API avec le framework FastAPI.

A partir d'un `id_vehicule`, cette API doit renvoyer l'ensemble des informations associée au véhicule recherché.

Les données proviendront de la table postgresql `vehicules`. Il faudra implémenter un système de cache via Redis si les données renvoyées sont identiques.

Exercice 1 : Installer et tester Redis via la CLI

1. Installer `redis`. Cette base de données ne fait pas partie du catalogue Onyxia. Pour l'installer, lancer un terminal puis lancer la commande :

```
kubectl run redis --image redis && kubectl expose pod/redis --port 6379
```

2. Installer la `cli` avec `sudo apt update && sudo apt install redis`
3. Se connecter à `redis` avec la commande `redis-cli -h redis` et créer une clé de type hash `AA-000-AA` avec comme sous clé `marque` `Mercedes`, `modele` `Classe A` et `annee` `2020` pour vérifier que Redis fonctionne.

Exercice 2 : lire les données de la base de données

1. Faire une fonction python qui lit les données de la table du précédent exercice avec la méthode `pd.read_sql` et qui renvoit le DataFrame. Appliquer avant de retourner le DataFrame la transformation suivante pour retirer les caractères spéciaux :

```
df.replace(r"\xa0", "", regex=True)
```

> Note : la fonction `pd.read_sql` nécessite un `engine` pour se connecter à la base de données

2. Pourquoi cette méthode de lecture de la base n'est pas recommandée dans le cadre d'une utilisation via une API ?

Exercice 3 : filtrer les données

1. Faire une fonction qui prend pour entrée un DataFrame et un `id_vehicule` et renvoyer la première ligne correspondante sous la forme d'un dictionnaire python avec la méthode `to_dict`.
2. Tester la fonction avec les `id_vehicule` suivants : `813953`, `8000000`. Que se passe t'il ?
3. Modifier cette fonction afin de renvoyer un dictionnaire vide si aucune ligne n'est trouvée

Exercice 4 : créer une API

1. A partir de la documentation de FastAPI créer l'api d'exemple dans un fichier `app.py` et la lancer avec la commande suivante dans un nouveau terminal linux :

```
fastapi dev app.py
```

> Il faudra au préalable installer fastapi

2. Tester un appel à cette api avec la commande `curl http://localhost:8000`

> curl est un outil pour lancer des requêtes HTTP

3. Modifier cette api pour prendre en paramètre le `id_vehicule` sous la forme `/vehicule/{id_vehicule}`

4. Ajouter les fonctions précédentes de recherche de données pour que l'API renvoit les informations du véhicule recherché. Ensuite, dans un autre terminal, tester le bon fonctionnement avec la commande :

```
curl http://localhost:8000/vehicule/813952
```

Exercice 5 : utiliser Redis avec Python

1. Installer le client python Redis

```
pip install redis
```

2. Dans un notebook, créer une connexion python à Redis et ajouter une clé `vehicule:813952` de type hash avec les informations correspondant à l'`id_vehicule` suivant : `813952`

3. Lire ensuite le hash précédent dans sa totalité et vérifier que les données sont correctes

4. Ajouter une expiration à la clé précédente de 60 secondes

Exercice 6 : intégrer Redis à une API

1. Pour chaque nouvel appel à l'API, ajouter les données dans Redis, seulement si elles n'existent pas

2. Pour chaque nouvel appel, ajouter une vérification de l'id dans Redis avant de lire la données dans la base de données

3. Ajouter un TTL de 60 secondes lors de l'ajout des données dans le cache