

Correction exercice : Intégration de Redis avec une API

L'objectif de cet exercice est d'implémenter une API avec le framework [FastAPI](#). A partir d'un `id_vehicule`, cette API renvoie l'ensemble des informations associées au véhicule. Les données proviennent de la table `vehicules`. Il faudra implémenter un système de cache si les données renvoyées sont identiques.

Quelques pistes

1. Utiliser `pd.read_sql` pour lire les données dans la base (*attention, non recommandé pour un vrai usage*).
2. Filtrer le jeu de données avec l'`id_vehicule` de l'appel API. Il est possible qu'il faille faire des transformations de données intermédiaire.
3. Retourner la ligne pandas sous forme d'un dictionnaire si le véhicule existe, sinon retourner un dictionnaire vide.
4. Pour chaque appel, ajouter les données dans `redis` si elles n'existent pas, avec un TTL de 60 secondes. Sinon renvoyer à la place les données du cache.

Méthode 1

1. Installer les différentes dépendances

```
pip install "fastapi[standard]" redis sqlalchemy
```

2. Créer un fichier `app.py` avec le contenu suivant (remplacer les `xxx` par les informations de votre environnement) :

```
import sqlalchemy
import pandas as pd
from fastapi import FastAPI
import redis

app = FastAPI()

con =
sqlalchemy.engine.create_engine("postgresql+psycopg2://postgres:4vr77ffm5
qllyqwm0hp@postgresql-xxx/defaultdb")
r = redis.Redis(host='redis', port=6379, decode_responses=True)

@app.get("/vehicule/{vehicule_id}")
def read_vehicule(vehicule_id: str):
    info = r.hgetall(vehicule_id)
    if "error" in info:
        return {}

    if "id_vehicule" in info:
```

```
        return info

df_temp = pd.read_sql("vehicules", con=con)
df_temp = df_temp.replace(r"\xa0", '', regex=True)
row = df_temp[df_temp["id_vehicule"] == vehicule_id]

if len(row):
    dict_row = row.iloc[0].to_dict()
    r.hset(vehicule_id, mapping=dict_row)
    r.expire(vehicule_id, 60)
    return dict_row

r.hset(vehicule_id, mapping={"error": "introuvable"})
r.expire(vehicule_id, 60)
return {}
```

Ce script permet de lancer une API qui écoute sur la route `/vehicule/{vehicule_id}` afin de renvoyer les informations du véhicule demandé.

- Vérifie que le véhicule n'est pas dans le cache, sinon renvoie les données depuis ce cache
- Lit la table `sql`
- Retourne la ligne correspondant à l'id recherché
- Ajoute les données dans le cache et lui applique un cache de 60 secondes

Quelques notes :

- l'id possède un formatage particulier, on applique donc une transformation à sa lecture
 - la méthode `hset` ne permet pas d'écrire un dictionnaire vide. Un contournement possible était donc de créer une clé spécifique si le véhicule n'existe pas et vérifier si cette clé est dans le cache.
 - Une autre méthode serait d'utiliser `set` et d'écrire le dictionnaire au format `string` dans redis, puis de le sérialisé en `json` lors de la lecture.
1. Dans un terminal lancer `fastapi dev app.py` pour lancer l'API
 2. Dans un autre terminal, tester un appel avec la commande `curl http://localhost:8000/vehicule/813952`. Si on rappelle ensuite l'API sur le même id, les données seront renvoyés en moins de temps

Méthode 2

1. Installer les différentes dépendances

```
pip install "fastapi[standard]" sqlalchemy fastapi_redis_cache
```

2. Créer un fichier `app2.py` avec le contenu suivant (remplacer les `xxx` par les informations de votre environnement) :

```
from fastapi import FastAPI, Request, Response
from fastapi_redis_cache import FastApiRedisCache, cache
```

```

from sqlalchemy.orm import Session
import sqlalchemy
import pandas as pd

con =
sqlalchemy.engine.create_engine("postgresql+psycopg2://postgres:xxx@postgr
esql-xxx/defaultdb")
app = FastAPI(title="FastAPI")

@app.on_event("startup")
def startup():
    redis_cache = FastApiRedisCache()
    redis_cache.init(
        host_url="redis://redis:6379",
        prefix="myapi-cache",
        response_header="X-API-Cache",
        ignore_arg_types=[Request, Response, Session]
    )

@app.get("/vehicule/{vehicule_id}")
@cache(expire=60)
def read_vehicule(vehicule_id: str, request: Request, response: Response):
    df_temp = pd.read_sql("vehicles", con=con)
    df_temp = df_temp.replace(r"\xa0", '', regex=True)
    row = df_temp[df_temp["id_vehicule"] == vehicule_id]
    if len(row):
        dict_row = row.iloc[0].to_dict()
        return dict_row
    return {}

```

Ce script permet de lancer une API qui écoute sur la route `/vehicule/{vehicule_id}` afin de renvoyer les informations du véhicule demandé. Il utilise un `middleware` utilisable par un système de décorateur pour ajouter dans le cache. La documentation est disponible [ici](#)

3. Dans un terminal lancer `fastapi dev app2.py` pour lancer l'API
4. Dans un autre terminal, tester un appel avec la commande `curl http://localhost:8000/vehicule/813952`. Si on rappelle ensuite l'API sur le même id, les données seront renvoyées en moins de temps