Vémock

I. Introduction

1. Contexte

Ce projet annuel se repose énormément sur le comportement des stations, car ce sont elles qui vont recharger les vélos et indique à l'utilisateur ou en prendra.

Puisque la mairie de Paris n'a pas souhaité nous financez pour mettre à disposition des stations dans la ville, il a fallu les simuler nous-mêmes pour avoir un comportement cohérent sur le site.

2. Application choisie

Nous avons développé Vemock pour répondre à ce besoin et simuler le comportement d'une station complètement autonome.

Cette application enverra régulièrement au serveur l'évolution d'une station autonome, à savoir le nombre d'emplacements vélos utilisés, le niveau de batterie, la puissance de charge, etc.

II. Focus sur l’application

1. État d’une station et configuration

Il y aura plusieurs paramètres à simuler dans le cycle de vie d'une borne, à savoir :

- La batterie
- Si la station est active
- La puissance de charge
- Le nombre d'emplacements de vélo utilisé

Ces paramètres doivent suivre une évolution cohérente donc certaines règles ont été mises en place afin d'apporter de la cohérence.

Comportement de la batterie

- Elle doit varier dans un intervalle de 0 à 100%
- Elle se décharge constamment, mais un peu moins la nuit.
- Elle dépend de la puissance de charge de la borne et un petit peu d'aléatoire
- Elle se décharge beaucoup plus vite s'il y a beaucoup de vélos sur la borne
- Si la batterie est en dessous de 15 pour 100, elle se rechargera plus facilement
- Dans le cas où la batterie tombe en dessous de 15 pour 100 et si la configuration le permet, la station passera en statut inactif

Puissance de charge

- Elle dépend d'un peu d'aléatoire
- La puissance de charge est faible le matin, très élevé en journée, puis à nouveau faible le soir.
- La puissance de charge est nulle la nuit

Nombre d’emplacements utilisés

- Le nombre d'emplacements utilisé est un peu aléatoire
- L'évolution du nombre d'emplacements utilisés doit être cohérentes par rapport au précédent état.

Configuration d’une station

Au démarrage, l'application va charger la configuration propre à la borne qu'elle doit simuler, à savoir :

- L'idée de la station
- Si elle a le droit de passer en statut inactif quand la batterie est trop basse
- Le nombre maximum d'emplacements vélo
- Le nom du fichier d'historique pour la reprise

- Le temps d'attente entre chaque envoi de données
- La puissance maximale de charge
- Le token d'authentification auprès du serveur
- Le lien pour accéder au serveur
- Si elle doit afficher des données de débogage pendant son exécution

Si ces données sont mises à jour et que la station redémarre, elle prendra en compte les changements.

2. Persistance de l’état et reprise

Si jamais une station s’éteint (l'application est coupée), elle ne doit pas repartir avec des valeurs initiales lorsqu'elle redémarre, mais bien avec les dernières données générées.

C'est pour cela qu’à chaque nouveau changement de statut de la borne, un fichier historique est mis à jour pour garder en mémoire le dernier statut connu de la borne.

Au démarrage de l'application, si ce fichier n'existe pas, alors l'application prend des valeurs initiales par défaut, sinon elle reprend là où elle s'était arrêtée.

3. Envoie des données

À chaque nouveau changement d’état, une métrique est envoyée sur le serveur API.

Cette requête contient :

- Le token d'autorisation de la borne dans l’en-tête qui permettrait au serveur d'identifier celle-ci
- Le pourcentage de batterie
- La puissance de charge
- Si elle est active
- Le nombre d'emplacements vélo utilisé

Nous utilisons le client HTTP standard fourni par la lib python.

```python
def submit_state(api_endpoint, state, jwt_token, debug=True):
    headers = {'Authorization': 'Bearer ' + jwt_token, 'Content-type': 'application/json'}
    data = {
        'battery': state.battery,
        'charging_power': state.charging_power,
        'active': state.active,
        'used_seats': state.used_seats,
    }
    if 'https://' in api_endpoint:
        api_endpoint = api_endpoint.replace('https://', '')
        api_splitted = api_endpoint.split('/', 1)
        conn = http.client.HTTPSConnection(api_splitted[0])
    else:
        api_endpoint = api_endpoint.replace('http://', '')
        api_splitted = api_endpoint.split('/', 1)
        conn = http.client.HTTPConnection(api_splitted[0])
    conn.request('POST', '/' + api_splitted[1], json.dumps(data), headers)
    if debug:
        print(conn.getresponse().read().decode())
```

4. Logging des actions

Si les logs sont activés ; à chaque changement d'état de la station ; il sera affiché en compte sol le nouvel état de la station ainsi que la requête envoyée.

Ces données sont également sauvegardées dans un fichier.

```
23/05/2021, 11:47:01 : {"battery": 49.836, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 4.0035, "max_charging_power": 30.0, "active": true}
23/05/2021, 11:47:02 : {"battery": 49.829, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 3.7738, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:35 : {"battery": 49.822, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 4.5596, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:37 : {"battery": 49.8168, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 7.2033, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:38 : {"battery": 49.8133, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 9.7208, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:40 : {"battery": 49.8124, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 10.0563, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:41 : {"battery": 49.8107, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 11.1512, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:43 : {"battery": 49.8099, "auto_off": false, "used_seats": 5, "max_seats": 10, "charging_power": 13.6019, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:15:44 : {"battery": 49.8099, "auto_off": false, "used_seats": 6, "max_seats": 10, "charging_power": 16.3688, "max_charging_power": 30.0, "active": true}
23/05/2021, 12:02:05 : {"battery": 49.8093, "auto_off": true, "used_seats": 6, "max_seats": 12, "charging_power": 17.6973, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:02:10 : {"battery": 49.8089, "auto_off": true, "used_seats": 6, "max_seats": 12, "charging_power": 17.7988, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:02:15 : {"battery": 49.8075, "auto_off": true, "used_seats": 6, "max_seats": 12, "charging_power": 18.1482, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:02:20 : {"battery": 49.8072, "auto_off": true, "used_seats": 6, "max_seats": 12, "charging_power": 19.6284, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:02:56 : {"battery": 49.8075, "auto_off": true, "used_seats": 7, "max_seats": 12, "charging_power": 19.7971, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:03:01 : {"battery": 49.8082, "auto_off": true, "used_seats": 7, "max_seats": 12, "charging_power": 20.0, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:03:06 : {"battery": 49.81, "auto_off": true, "used_seats": 7, "max_seats": 12, "charging_power": 20.0, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:07:05 : {"battery": 49.8099, "auto_off": true, "used_seats": 7, "max_seats": 12, "charging_power": 20.0, "max_charging_power": 20.0, "active": true}
23/05/2021, 12:07:10 : {"battery": 49.8114, "auto_off": true, "used_seats": 7, "max_seats": 12, "charging_power": 20.0, "max_charging_power": 20.0, "active": true}
29/06/2021, 16:48:38 : {"battery": 49.7456, "auto_off": false, "used_seats": 7, "max_seats": 10, "charging_power": 15.546, "max_charging_power": 30.0, "active": true}
29/06/2021, 16:48:49 : {"battery": 49.6835, "auto_off": false, "used_seats": 6, "max_seats": 10, "charging_power": 18.6824, "max_charging_power": 30.0, "active": true}
29/06/2021, 16:48:59 : {"battery": 49.6115, "auto_off": false, "used_seats": 6, "max_seats": 10, "charging_power": 13.7796, "max_charging_power": 30.0, "active": true}
```

5. Générateur du docker-compose

Pour simuler le comportement de 100 stations, le fichier docker compose **docker-compose.yml** fais plus de 2400 lignes. Bien sûr, il serait trop long d'écrire toutes ces lignes à la main...

Nous avons donc mis au point un script python qui permet à partir d'un fichier de configuration JSON beaucoup moins gros de générer notre **docker-compose.yml** final.

Voici un exemple des fichiers de configuration avec seulement 2 stations :

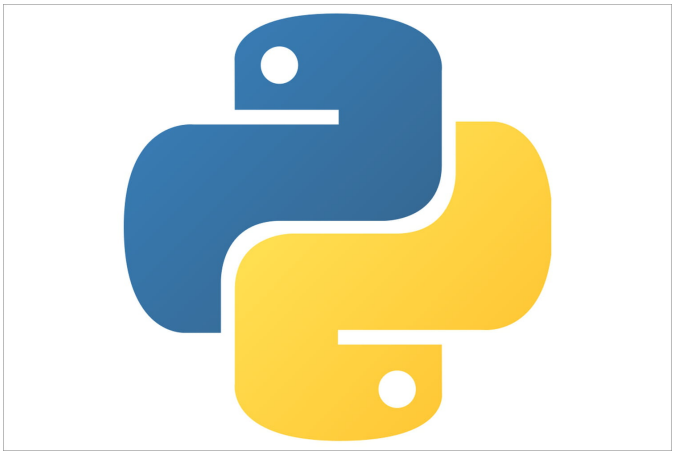
config.json | docker-compose.yml

```
{
  "config": {
    "frequency_wait": 5,
    "history_path": "history.json",
    "max_log_size": "30m",
    "max_log_file": 10,
    "api_endpoint": "[redacted]",
    "debug":  "True"
  },
  "station_list": [
    {
      "station_id": 1,
      "auto_off": "True",
      "max_seats": 12,
      "max_charge": 20,
      "jwt_token": "[redacted]iI"
    },
    {
      "station_id": 2,
      "auto_off": "True",
      "max_seats": 6,
      "max_charge": 20,
      "jwt_token": "[redacted]v_hY"
    }
  ]
}
```

```yaml
version: '3.8'
services:
  vemock_station_1:
    environment:
      - station_id=1
      - auto_off=True
      - max_seats=12
      - history_path=history.json
      - frequency_wait=5
      - max_charge=20
      - jwt_token=eyJ[redacted]
      - api_endpoint=https://ap[redacted]
      - debug=True
    volumes:
      - vemock_station_1:/usr/src/app/history
    networks:
      - vemock_network
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "30m"
        max-file: "10"
    image: vemock
  vemock_station_2:
    environment:
      - station_id=2
      - auto_off=True
      - max_seats=6
      - history_path=history.json
      - frequency_wait=5
      - max_charge=20
      - jwt_token=eyJhb[redacted]hY
      - api_endpoint=https[redacted]
      - debug=True
    volumes:
      - vemock_station_2:/usr/src/app/history
    networks:
      - vemock_network
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "30m"
        max-file: "10"
    image: vemock
volumes:
  vemock_station_1:
    driver: local
  vemock_station_2:
    driver: local
networks:
  vemock_network:
    name: vemock_network
    driver: overlay
    attachable: true
```

III. Choix d’implémentations

1. Langage choisi

Pour ce projet, nous avons choisi d'utiliser le langage python, car c'est selon nous celui qui répondait le mieux à notre besoin. Il possède beaucoup de librairies permettant de parser et traiter des données, ce qui nous a permis d'aller relativement vite.	

2. Contrainte de performance

Puisque cette application est destinée à tourner en plusieurs exemplaires sur un Raspberry, il faut que celui-ci soit le moins gourmand possible.

C'est pourquoi nous n'utilisons pas de librairies externes, mis à part **dotenv** dans l'environnement de développement.

Avec toutes les optimisations faites, l'application ne prend pas plus de 10 Mo de mémoire vive pendant son exécution.

IV. Bilan du projet

1. Problèmes rencontrés

Librairie Requests trop lourde

Au tout début, nous n'utilisions pas le client natif python, mais une librairie se nomment **Requests**.

Lors de l'exécution de l'application, nous constations que celle-ci prenait énormément de mémoire. Après beaucoup de vérifications, nous avons découvert que cela venait de cette librairie lorsqu'elle était importée.

Nous avons donc dû faire un changement pour utiliser plutôt la librairie native de python, qui est un peu plus verbeuse, mais moins consommatrice.

Équilibrage des métriques

L'algorithme définissant l'état de la station n'a pas été parfait du premier coup et nous faisions face à des données qui n'étaient pas cohérentes (batterie toujours à 100%, ne charge nulle, capacité de vélo dépassé ...).

L’application a ainsi subi plusieurs évolutions pour que le comportement soit le plus cohérent possible.

Fuites de mémoires sur la durée

En plus de la librairie **Requests** que nous avons enlevé, nous constations des fuites de mémoire lors de l'exécution prolongée de l'application.

Cela venait du fait qu'à chaque requête où log, l'application gardé en cache certaines données et ne les libérait pas tout de suite.

Afin de garder une consommation de mémoire stable, nous manipulons manuellement le **Garbage Collector** de python pour le forcer à se vider à chaque changement d'état.

2. Conclusion

Bien que ce projet ne soit absolument pas demandé à la base, il était essentiel par rapport au sujet que nous avons choisi.

Nous étions totalement dans l'inconnu lorsque nous avons développé cette application et il a fallu recommencer plusieurs fois, car nous n'avons pas compris tout de suite les problèmes que nous avons rencontrés.

Cependant, nous sommes très satisfaits du résultat et il a enrichi énormément le projet final.

Revision #2
Created 27 September 2022 17:38:59 by Noé Larrieu-Lacoste
Updated 27 September 2022 19:00:24 by Noé Larrieu-Lacoste