TP Messenger

UE12 P24 - Python

Objectif

Être capable de s’envoyer des messages via un serveur.

Découpage en étapes

Constat : c’est compliqué -> on y va par étapes

1. On simule le serveur grâce un objet Python
2. On remplacera ensuite l’objet pas à pas

Simuler le serveur

Quelles données sur le serveur ?

- Des utilisateurs
- Des groupes de discussion
- Des messages

On commence par la représentation la plus simple possible : dictionnaires et listes

Exercice

À partir du fichier `messenger.py` :

1. Ajouter un menu qui permet d'afficher les utilisateurs
2. Ajouter un menu qui permet d'afficher les groupes
3. Ajouter une option pour afficher les messages d'un groupe

> **Note**
>
> À chaque étape :
>
> - Faire un commit
> - Refactorer (améliorer l'organisation du code)

Exercice

4. Permettre d’ajouter des utilisateurs
5. Permettre d’ajouter des groupes

Problème : comment sauvegarder les données entre 2 lancements ?

Sauvegarde de données

Plusieurs solutions :

1. Une base de données

   -> La bonne solution en temps normal

2. Un fichier

   -> Plus simple, donc adapté pour une solution transitoire (en attendant la connexion serveur)

Représenter des données structurées

Plusieurs formats existent :

- CSV
- JSON
- YAML

JSON - Exemple

Un objet : ici un utilisateur

```json
{
    "id": 1,
    "name": "My username"
}
```

Une liste : ici des noms

```json
["Alice", "Bob", "Charlie"]
```

La combinaison : une liste d'utilisateurs

```json
[
    {"id": 18, "name": "Alice"},
    {"id": 5, "name": "Bob"}
]
```

> **Note**
>
> Les espaces et sauts de ligne ne servent qu'à la lisibilité

Json - lecture en Python

- `json.loads` : pour *parser* une chaîne de caractères
- `json.load` : pour *parser* un fichier

```
1 import json
2
3 my_variable = json.loads('{"id": 35, "name": "Alice"}')
```

Quel sera le type de `my_variable` ?

`dict`

Exercice - Json

1. Déplacer les données du dictionnaire `server` dans un fichier JSON. Vérifier que tout fonctionne.

> **Note**
>
> Si les emojis ne s'affichent pas bien, on verra plus tard pourquoi

2. Ajouter une fonction de sauvegarde du fichier JSON.

> **Note**
>
> Regarder le contenu du fichier json après modification

Modélisation

Vos données sont représentées par des dictionnaires, ce qui est peu maintenable. Si vous souhaitez remplacer l'attribut `name` des `users` par 2 attributs `firstname` et `lastname` :

- vous devez chercher toutes les occurrences de `['name']`
- bon courage pour savoir si `d['name']` est un `user` ou un `channel`

Modélisation - `User`, `Channel`, `Message`

1. Créer une classe `User`
2. *refactorer* votre code pour remplacer tous vos dictionnaires `user` par des classes
3. Faire de même avec `Channel` et `Message`

> **Tip**
>
> - Pensez à bien typer toutes vos fonctions !
> - Ne changez pas les `User`, les `Channel`, et les `Message` d'un coup : ce sera plus difficile de trouver d'éventuelles erreurs

Modélisation

Objectif :

- on veut communiquer sur le réseau
- on va découpler le code lié au client et le code lié au serveur, pour progressivement remplacer le code du serveur

Modélisation - `Server`

Le `server` est toujours un dictionnaire, les fonctions `load_server` et `save_server` ne lui sont pas rattachées.

1. Transformer le dictionnaire `server` en instance d'une nouvelle classe `Server`
2. Transformer les fonctions `load_server` et `save_server` en méthodes `Server.load` et `Server.save`

Modélisation - `Client`

C’est mieux, mais le `server` est toujours une variable globale directement utilisée dans les fonctions.

1. Créer une classe `Client`
2. Transformer toutes les fonctions d’affichage en méthodes de la classe `Client`
3. Transformer la variable globale `server` en variable de la classe `Client`
4. Dans `Client`, remplacer tous les appels à `server` par des appels à `self.server`

Modélisation - Récapitulatif

Les seules lignes de code qui doivent rester en dehors des classes sont :

```
1 SERVER_FILE_NAME = 'server-data.json'
2 server = Server(SERVER_FILE_NAME)
3 server.load()
4 client = Client(server)
5 client.welcome_screen()
```

Modélisation - Structure

Votre code est mieux structuré : plutôt que de réfléchir à l'agencement de beaucoup de fonctions au même niveau, vous pouvez penser aux interactions entre un `Client` et un `Server`.

Vous allez facilement pouvoir remplacer ce `Server` local par un `Server` qui communique sur le réseau.

Paramètres de script

Vous êtes presque prêts à faire communiquer votre script avec un serveur. Vous aurez donc bientôt 2 choix : utiliser un serveur local ou un serveur distant.

Le moyen classique de gérer ce choix est de donner un argument à votre script :

```
1 python messenger.py --local
2 python messenger.py --remote
```

Paramètres de script

Pour commencer, cherchez sur internet comment passer le chemin du fichier json en paramètre :

```
1 python messenger.py --server-file server-data.json
```

Puis faites une pull request (PR) à un·e camarade pour coder cette fonctionnalité dans son projet.

> ⚠ **Warning**
>
> Pour pouvoir intégrer la PR de vos camarades sereinement, commencez par mettre vos modifications en cours sur une nouvelle branche

Découpage en fichiers

Tout votre code est dans le même fichier, qui devient trop gros.

Séparer votre code en 4 fichiers :

- model.py : contient les classes `Channel`, `Message`, `User`
- server.py : contient la classe `Server`
- client.py : contient la classe `Client`
- messenger.py : contient le reste

Héritage

On veut avoir 2 types de serveurs différents :

- `LocalServer` est notre serveur actuel, basé sur un fichier json
- `RemoteServer` va être notre interface avec le serveur web

Mais le `Client` n'a pas besoin de connaître la différence : il va continuer d'utiliser un `Server` abstrait.

Héritage - Exercice

1. Copier votre fichier `server.py` dans un fichier `localserver.py`
2. Dans `server.py` :
   - remplacez les implémentations des méthodes par des messages d'erreur
   - enlevez les méthodes inutiles
3. Dans `localserver.py` : remplacez `Server` par `LocalServer` et ajoutez des `@override` sur les méthodes
4. Dans les autres fichiers : faut-il garder `LocalServer` ou `Server` ?

Web

Vous êtes prêt·e·s à développer le lien avec le serveur distant.

1. Créer une classe `RemoteServer`
2. Ajouter une option `--url` à votre script. Lorsqu'elle est utilisée, créer un `RemoteServer` au lieu d'un `LocalServer`
3. Dans votre navigateur, aller sur http://vps-cfefb063.vps.ovh.net/channels
4. Implémenter la méthode `RemoteServer.get_channels`
5. Implémenter les autres méthodes
    - Implémenter une des méthodes via une `PR` chez la personne qui vous en a déjà fait une

Web - URL utiles

Utilisateurs :

- Liste : `/users`
- Détails : `/users/$id`
- Création : POST `/users/create`, paramètre `name`

Channels :

- Liste : `/channels`
- Détails : `/channels/$id`
- Création : POST `/channels/create`, paramètre `name`
- Ajout utilisateur : POST `/channels/$id/join`, paramètre `user_id`

Web - URL utiles

Channels :

- Membres : `/channels/$id/members`
- Messages : `/channels/$id/messages`
- Envoyer un message : POST `/channels/$id/messages/post`