DESSIS-GEAY Tristan IAS-M1-DS
FEIST Capucine

Projet : Simulation d'un système de réservation de vols

Les classes :

- Classe Place :
    - Attributs :
        - numéro : numéro de la place
        - reservee : booléen pour savoir si la place est réservée ou non
    - Méthodes :
        - reserver() : réserve la place si elle est toujours disponible
        - annuler() : annule la réservation si il y'en a une
        - __str__ () : renvoie un texte pour représenter l'objet

- Classe Vol :
    - Attributs :
        - n-vol : numéro du vol
        - ville_depart : ville de départ du vol
        - ville_arrivee : ville d'arrivée du vol
        - sieges_total : nombre total de sièges dans l'avion
        - sieges_dispo : nombre de sièges disponibles dans l'avion
        - liste_places = liste d'objets Place
    - Méthodes :
        - reserver_siege() : réserve un siège disponible et met à jour la disponibilité
        - __str__ () : renvoie le n° de vol avec le nombre de sièges disponibles

- Classe Utilisateur :
    - Attributs :
        - id : identifiant unique du passager
        - prenom : prénom du passager
        - nom : nom du passager
        - age : âge du passager
        - reservations : réservations effectuées par le passager

- Classe Reservation :
    - Attributs :
        - id : id unique de la réservation
        - id_utilisateur : id de l'utilisateur associé à la réservation
        - id_vol : id unique du vol
        - id_place : id unique de la place
        - date : date du vol

- Classe Ville :
    - Attributs :
        - id : id de la ville
        - nom : nom de la ville
    - Méthode :
        - __str__ () : renvoie le nom de la ville

- Classe Dataset :
    - Attribut :
        - path : chemin du fichier json
    - Méthodes :
        - sauvegarder() : permet de sauvegarder les données dans le json
        - create_objects() : créée les différents objets (utilisateurs, vols etc)
        - create_user() : permet de créer un utilisateur et de l'ajouter au fichier json
        - create_ville() : permet de créer une ville et de l'ajouter au fichier json
        - create_vol() : permet de créer un vol et de l'ajouter au fichier json
        - create_reservation() : permet de créer une réservation et de l'ajouter au fichier json
        - drop_reservation() : permet de supprimer une réservation et mettre à jour les différents états (place, utilisateur etc) dans le fichier json
        - trouver_vol() : permet à l'utilisateur de rechercher un vol et d'obtenir des informations dessus (ville de départ, ville d'arrivée etc). En cas d'erreur, renvoie un message disant que le numéro de vol recherché n'existe pas.
        - lister_reservations() : permet de lister les réservations effectuées par un utilisateur avec les détails (ville de départ, id vol etc). En cas d'erreur, renvoie un message en disant que l'id utilisateur n'existe pas.

Les tests :

Pour garantir la fiabilité et le bon fonctionnement de notre système de réservation de vol, nous avons développé une suite de tests automatisés, que nous avons placée dans un fichier intitulé Tests.py. Ces tests sont exécutés via une fonction principale, main(), qui orchestre l'ensemble des vérifications.

Les tests sont conçus pour évaluer plusieurs aspects cruciaux du système, notamment :

- Réservation de Sièges :
    - La fonction check_place_count_decreases(vol) teste la fonctionnalité de réservation. Elle vérifie que le nombre de sièges disponibles diminue de manière appropriée lorsqu'un utilisateur réserve un siège.

- Association des Réservations à des Utilisateurs :
    - Avec check_reservation_associate_user(ds, id_reservation), nous nous assurons que chaque réservation est correctement liée à un utilisateur valide, garantissant ainsi l'intégrité des données.

- Annulation de Réservations :
    - La fonction check_place_count_increase(ds, vol, id_reservation) teste que le nombre de sièges disponibles augmente après l'annulation d'une réservation, confirmant ainsi que le système gère correctement les places libérées.

- Gestion des Réservations d'Utilisateurs :
    - check_reservation_cancel(ds, user, vol) vérifie que le nombre total de réservations d'un utilisateur est réduit lorsque celui-ci annule une réservation, s'assurant ainsi que notre système reste en phase avec les attentes des utilisateurs.

- Persistance des Données :
    - Enfin, check_saving_loading(ds) examine la fonctionnalité de sauvegarde et de chargement des données. Cette fonction teste que les données peuvent être écrites dans un fichier JSON et rechargées ultérieurement sans perte d'information.

À la fin de l'exécution des tests, le code affiche un message 'OK' si toutes les vérifications passent sans erreur. Dans notre cas, ce résultat indique que toutes les différentes méthodes de notre système fonctionnent correctement, confirmant ainsi la robustesse et l'efficacité de l'implémentation.

Persistance des données :

Nous avons choisi d'utiliser un JSON afin de stocker nos données.
Il est composé de 4 objets, à savoir l'utilisateur, les réservations, le vol et les villes qui contiennent chacun plusieurs données.
Nous les récupérons avec `json.load(json_file)`

Une fois nos données rentrées dans nos différents objets, nous les envoyons de le fichier JSON avec `json.dump(new_data, json_file)`

Les avantages de JSON sont sa simplicité, il est facile de lire et d'écrire, de plus, il ne nécessite pas de configuration complexe à l'inverse du cas d'une base de données.
Il nous permet de stocker des données structurées et de conserver leur état entre les exécutions de l'application.