Dépendances

- `socket`: Fournit des outils pour travailler avec des connexions réseau.
- `ssl`: Permet la sécurisation des sockets avec SSL/TLS.
- `json`: Pour la gestion des données au format JSON.
- `os`: Utilisé pour la gestion des fichiers et des répertoires.
- `uuid`: Pour générer des identifiants de session uniques.
- `zipfile`: Permet de compresser les fichiers.
- `datetime`: Pour travailler avec les dates et heures.
- `base64`: Pour l'encodage et le décodage des fichiers binaires.
- `logging`: Pour enregistrer les logs du serveur.
- `secure_storage`: Contient la classe `SecureFileStorage` pour le stockage sécurisé des fichiers.

Variables de Configuration

Les variables suivantes sont configurées pour le serveur :

- `server_host`: L'adresse IP du serveur (par défaut `0.0.0.0` pour accepter toutes les connexions).
- `server_port`: Le port d'écoute du serveur, configurable via la variable d'environnement `PORT`, sinon par défaut à `5000`.
- `server_cert`: Le chemin vers le certificat SSL du serveur.
- `server_key`: Le chemin vers la clé privée SSL du serveur.
- `ca_cert`: Le chemin vers le certificat de l'autorité de certification pour vérifier les clients.

Structure des Utilisateurs et Sessions

- `users`: Un dictionnaire contenant les noms d'utilisateurs et mots de passe pour l'authentification.
- `sessions`: Un dictionnaire stockant les sessions actives des utilisateurs, identifiées par des UUID.
- `secure_storage`: Un dictionnaire qui associe chaque session à une instance de `SecureFileStorage`.

Fonctions

`create_secure_socket()`

Crée et retourne un socket sécurisé en utilisant SSL/TLS. Le serveur écoute sur `server_host` et `server_port`.

Retour:

- `secure_socket`: Un objet `SSL socket` prêt à accepter des connexions.

`parse_headers(request_data)`

Extrait les en-têtes HTTP de la demande brute (`request_data`).

Retour:

- `headers`: Un dictionnaire contenant les en-têtes HTTP extraits.

`send_response(client_socket, status_code, body)`

Envoie une réponse HTTP au client avec le code de statut et le corps spécifié. Le corps est sérialisé en JSON.

Paramètres:

- `client_socket`: Le socket du client.
- `status_code`: Le code de statut HTTP à envoyer.
- `body`: Le corps de la réponse au format JSON.

`handle_login(client_socket, request_data)`

Gère les demandes de connexion des utilisateurs. Si les identifiants sont corrects, une session est créée et l'utilisateur est authentifié.

Paramètres:

- `client_socket`: Le socket du client.
- `request_data`: Les données de la requête HTTP contenant les informations de connexion (nom d'utilisateur et mot de passe).

`handle_file_upload(client_socket, headers, request_data)`

Gère les demandes de téléchargement de fichiers. Le fichier est stocké de manière sécurisée et crypté dans un fichier archive.

Paramètres:

- `client_socket`: Le socket du client.
- `headers`: Les en-têtes HTTP extraits de la requête.
- `request_data`: Les données de la requête HTTP contenant le fichier à télécharger.

`handle_get_files(client_socket, headers)`

Retourne une liste des archives disponibles (fichiers cryptés) pour l'utilisateur authentifié.

Paramètres:

- `client_socket`: Le socket du client.
- `headers`: Les en-têtes HTTP extraits de la requête.

`handle_download_file(client_socket, headers)`

Gère les demandes de téléchargement de fichiers. Si l'utilisateur est autorisé, le fichier est récupéré et son contenu est envoyé en réponse sous forme de chaîne encodée en base64.

Paramètres:

- `client_socket`: Le socket du client.
- `headers`: Les en-têtes HTTP extraits de la requête.

`handle_client(client_socket)`

Gère une connexion client. En fonction de la requête HTTP, cette fonction appelle les gestionnaires appropriés (connexion, téléchargement de fichier, etc.).

Paramètres:

- `client_socket`: Le socket du client.

`start_server()`

Démarre le serveur, crée un socket sécurisé et attend des connexions clientes. Lorsqu'une connexion est établie, elle est traitée par `handle_client()`.

Flux de Travail

1. **Création du serveur** : Lors du démarrage, un socket sécurisé est créé avec les certificats SSL. Le serveur écoute les connexions entrantes.
2. **Connexion de l'utilisateur** : L'utilisateur envoie ses identifiants via une requête POST `/login`. Si l'authentification réussit, une session est créée et un identifiant de session est retourné.
3. **Téléchargement de fichiers** : L'utilisateur peut télécharger des fichiers via une requête POST `/upload-file`. Les fichiers sont cryptés et stockés de manière sécurisée.
4. **Liste des archives** : L'utilisateur peut consulter la liste des fichiers cryptés stockés via une requête GET `/get-files`.
5. **Téléchargement d'un fichier** : L'utilisateur peut télécharger un fichier crypté via une requête GET `/download-file`.

Gestion des Erreurs

Les erreurs sont gérées par des blocs `try-except`, et les messages d'erreur sont enregistrés à l'aide de la bibliothèque `logging`. Si une erreur se produit, une réponse HTTP 500 (Erreur interne du serveur) est envoyée.

Sécurité

- Le serveur utilise SSL/TLS pour sécuriser les connexions.
- Les fichiers sont cryptés après leur téléchargement.
- Les utilisateurs sont authentifiés via un nom d'utilisateur et un mot de passe stockés en texte clair (non recommandé pour un environnement de production).