DESCRIPTION PROJET API

1. Description du Projet

L'objectif de ce projet est de développer la partie "Back-end" d'une application de gestion d'étudiants, en utilisant une architecture RESTful. Cette application permet d'exécuter les opérations de CRUD (Create, Read, Update, Delete) sur une base de données contenant des informations d'étudiants.

1.1 Diagramme de Cas d'Utilisation

(Insérez ici le diagramme de cas d'utilisation montrant les interactions des utilisateurs avec l'API pour les opérations CRUD sur la base de données des étudiants)

2. Description de l'API

L'API a été conçue en utilisant Flask pour router les requêtes HTTP vers des fonctions dédiées, lesquelles interagissent avec une base de données MySQL. Voici une présentation détaillée des routes principales avec les paramètres et codes de retour possibles, décrite en utilisant le format Swagger.

2.1 Routes et Requêtes HTTP

1. **Obtenir tous les étudiants**
   - **Méthode** : GET
   - **URL** : `/v1/etudiants/`
   - **Réponses** :
     - 200 : Retourne la liste de tous les étudiants.
     - 400 : Requête invalide.
     - 401 : Accès non autorisé.
     - 500 : Erreur serveur (connexion à la base échouée).
2. **Obtenir un étudiant par ID**
   - **Méthode** : GET
   - **URL** : `/v1/etudiants/{id}`
   - **Paramètres** : `id` de l'étudiant
   - **Réponses** :
     - 200 : Détails de l'étudiant en JSON.
     - 400 : Requête invalide.
     - 404 : ID de l'étudiant invalide.
     - 500 : Erreur de connexion à la base.
3. **Ajouter un étudiant**
   - **Méthode** : POST
   - **URL** : `/v1/etudiants/`
   - **Paramètres** : Détails de l'étudiant en JSON (dans le body)

    - **Réponses** :
        - 201 : Étudiant ajouté avec succès.
        - 401 : Accès non autorisé.
        - 500 : Erreur serveur.
4. **Modifier un étudiant**
    - **Méthode** : PUT
    - **URL** : `/v1/etudiants/{id}`
    - **Paramètres** : `id` de l'étudiant, détails mis à jour de l'étudiant en JSON (body)
    - **Réponses** :
        - 200 : Modification réussie.
        - 401 : Accès non autorisé.
        - 404 : ID de l'étudiant invalide.
        - 500 : Erreur serveur.
5. **Supprimer un étudiant**
    - **Méthode** : DELETE
    - **URL** : `/v1/etudiants/{id}`
    - **Paramètres** : `id` de l'étudiant
    - **Réponses** :
        - 200 : Suppression réussie.
        - 401 : Accès non autorisé.
        - 404 : ID de l'étudiant invalide.
        - 500 : Erreur serveur.

---

3. Récapitulatif des Versions de l'API

- **Version 1 (v1)** :
    - Fonctionnalités CRUD de base.
    - Utilisation des f-strings pour générer les requêtes SQL.
    - Routes versionnées avec le préfixe `/v1/`.
- **Version 2 (v2)** :
    - Gestion améliorée des erreurs avec try/except.
    - Création d'un utilisateur externe pour une utilisation en réseau local.
    - Début de l'intégration de la gestion des privilèges.
- **Version 3 (v3)** :
    - Ajout de la table `user` et gestion des connexions avec identifiant et mot de passe.
    - Mise en place d'une route pour l'authentification (`/login` avec méthode POST).
    - Refactorisation vers une approche modulaire et orientée objets avec la classe `Database` dans un fichier séparé (`db.py`).

---

4. Diagramme de Classes pour la Classe `Database`

La classe `Database` est le cœur de la gestion des interactions avec la base de données. Elle encapsule les fonctions de connexion et d'exécution des requêtes SQL pour l'API. Voici le diagramme de classes généré avec Pyreverse.

(Insérez ici le diagramme de classes Pyreverse pour la classe Database)

---

5. Fiche des Tests de l'API (Postman)

Les tests de l'API ont été réalisés par un autre étudiant, utilisant Postman pour valider le bon fonctionnement de chaque route.

Test ID	Méthode	Route	Paramètres	Résultat Attendu	Résultat Obtenu	Statut
T1	GET	/v1/etudiants/	N/A	200 - Liste des étudiants	200	OK
T2	GET	/v1/etudiants/{id}	id	200 - Détails de l'étudiant	200	OK
T3	POST	/v1/etudiants/	Détails étudiant (body)	201 - Ajout réussi	201	OK
T4	PUT	/v1/etudiants/{id}	id, Détails (body)	200 - Modification réussie	200	OK
T5	DELETE	/v1/etudiants/{id}	id	200 - Suppression réussie	200	OK
T6	POST	/v3/login/	Identifiant/mot de passe	200 - Authentification réussie	200	OK

6. Conclusion et Améliorations en Cybersécurité

Cette API répond aux fonctionnalités de gestion d'étudiants et est suffisamment flexible pour être étendue. Cependant, des améliorations peuvent être apportées pour renforcer la sécurité :

1. **Protection contre les injections SQL** : Actuellement, l'API utilise les f-strings pour générer les requêtes SQL, ce qui la rend vulnérable aux injections. Utiliser des requêtes préparées avec des paramètres permettrait de réduire ce risque.
2. **Authentification renforcée** : L'ajout d'un mécanisme de token (JWT) pour la gestion de sessions sécurisées et la mise en place d'une authentification à deux facteurs (2FA) offrirait une sécurité supplémentaire.
3. **Détection d'injections XSS** : Contrôler et filtrer les données d'entrée pour empêcher les attaques par injection XSS, qui pourraient compromettre la sécurité des données.
4. **Journalisation et surveillance** : Mettre en place un système de journalisation pour enregistrer les tentatives de connexion et autres opérations sensibles, afin de détecter des comportements suspects.
5. **Automatisation des tests de sécurité** : Intégrer un outil de scan de sécurité dans le processus CI/CD pour détecter les vulnérabilités courantes, comme les failles d'injection et les erreurs de configuration.