Évaluation J1, SecuVault

Le sujet

Vous allez construire **SecuVault**, une application web de coffre-fort de mots de passe partagés entre équipes.

Fonctionnalités attendues :

- Un utilisateur se connecte avec login et mot de passe.
- Un utilisateur appartient à une ou plusieurs équipes.
- Un secret (nom + valeur) appartient à une équipe. Il est chiffré en base.
- Un utilisateur peut lister les secrets des équipes dont il est membre.
- Un utilisateur peut révéler la valeur d'un secret, uniquement s'il y a accès.
- Un utilisateur peut créer un secret pour une équipe dont il est membre.
- Un utilisateur peut faire tourner un secret (remplacer sa valeur).

Stack imposée : Python, FastAPI, Jinja2, SQLite. Bibliothèques autorisées : `cryptography` pour le chiffrement symétrique, `bcrypt` pour le hash des mots de passe.

Jeu de données à provisionner : trois utilisateurs (`alice`, `bob`, `charlie`), deux équipes (`devops`, `marketing`). Alice dans `devops`, Bob dans `devops` et `marketing`, Charlie dans `marketing`.

Ce qui est évalué

Le respect strict du découpage en trois couches vu aujourd'hui. Le placement correct des règles métier dans la bonne couche. L'application du pattern MVC dans la couche présentation. La gestion de la concurrence sur la rotation d'un secret. La résistance à trois scénarios d'attaque.

Ce n'est pas un exercice d'ingénierie crypto. On ne vous demande pas d'inventer des algorithmes. On vous demande de bien placer les responsabilités.

Le scénario de concurrence à gérer

Deux administrateurs de la même équipe font la rotation du même secret au même moment. Sans protection, la dernière écriture écrase silencieusement la première, et un des deux admins croit avoir changé le mot de passe alors qu'il ne l'a pas fait. Votre application doit détecter ce conflit et avertir l'utilisateur concerné plutôt que de perdre une écriture. La preuve se fait avec deux navigateurs ouverts en parallèle.

BONUS : Les scénarios d'attaque à bloquer

Votre application doit résister aux trois scénarios suivants. À chacun, documentez dans un fichier `ATTACKS.md` le test mené, le résultat attendu et le résultat obtenu.

1. Un utilisateur authentifié accède directement, via l'URL, à un secret d'une équipe dont il n'est pas membre.

2. Un utilisateur authentifié envoie une requête de création de secret pour une équipe dont il n'est pas membre.
3. Un utilisateur authentifié teste des identifiants de secrets au hasard pour deviner ce qui existe.

Règles de l'évaluation

Le code est à rendre sous forme d'une arborescence claire, conforme au découpage en trois couches. Un `README.md` court explique comment lancer l'application et les tests. Vous pouvez poser des questions de clarification sur le sujet, pas sur la solution. Une règle métier trouvée dans un template ou dans une route est comptée comme une erreur d'architecture. Une règle d'accès absente du domaine est comptée comme une faille de sécurité.

Vous disposez de 6 heures.