Rapport Mini Projet (Flask) - UserVault

Auteur : Mohamed Omar Azzaoui
Encadrant: Dr. Othmane Bakkali Yedri

Date : 28 mars 2025

1. Introduction

Le dépôt **UserVault** présente une application web conçue en Python, visant à gérer de manière sécurisée l'authentification et la gestion des utilisateurs. Construit avec le framework Flask, ce projet se distingue par l'implémentation d'une architecture MVC (Modèle-Vue-Contrôleur) qui sépare clairement les responsabilités du code. L'objectif est de faciliter la maintenance et l'évolution du projet tout en garantissant une sécurité renforcée pour la gestion des données sensibles.

2. Structure du Projet

USERVAULT/

├── .env

├── .gitignore

├── requirements.txt

├── config.py

├── extensions.py

├── app/

├── controllers/

├── models/

├── services/

├── static/

├── templates/

├── migrations/

├── tests/

Le dépôt est organisé de manière à respecter les principes de modularité et de séparation des préoccupations. Voici les principaux dossiers et fichiers observés dans le projet :

- **/app**
  Ce dossier contient l'ensemble des composants de l'application :
  - **/models** : Contient les classes de modèles représentant les entités de la base de données (eg.User, Diary).
  - **/controllers** : Regroupe la logique métier et la gestion des routes. Les fichiers de contrôleurs gèrent les interactions utilisateur (par exemple, la connexion, l'inscription et la gestion des profils).
  - **/templates** : Fichiers HTML utilisant Jinja2 pour générer dynamiquement les pages de l'application.
  - **/static** : Contient les ressources statiques telles que les feuilles de style CSS et les scripts JavaScript.
  - **/services**
    Logique métier complexe ou interactions externes (ex: API tierces, calculs).
- **config.py**
  Fichier de configuration centralisant les paramètres de l'application (clé secrète, URI de la base de données, etc.).
- **run.py**
  Point d'entrée de l'application, qui initialise Flask, charge la configuration et démarre le serveur.
- **`extensions.py`**
  Initialisation des extensions Flask (ex: base de données, authentification).

Cette organisation permet de maintenir un code clair et évolutif, facilitant ainsi l'ajout de nouvelles fonctionnalités.

3. Architecture MVC et Analyse des Composants

L'application UserVault adopte le modèle MVC pour assurer une séparation nette entre la gestion des données, la logique métier et la présentation.

3.1 Le Modèle (Model)

- **Description :**
  Les modèles définissent la structure des données et les règles de gestion associées. Par exemple, le fichier user.py dans le dossier **/models** décrit le modèle User avec des attributs essentiels comme l'identifiant, le nom d'utilisateur, l'email et le mot de passe haché.
- **Fonctionnalités :**
  - **Gestion des mots de passe :** Des méthodes telles que set_password() et check_password() permettent de hacher et de vérifier les mots de passe à l'aide d'outils comme Werkzeug.
  - **Interaction avec la base de données :** Grâce à SQLAlchemy, le modèle facilite les opérations CRUD (création, lecture, mise à jour, suppression) sans avoir à écrire de requêtes SQL brutes.
- **Avantages :**
  Une gestion centralisée et sécurisée des données, conforme aux bonnes pratiques de conception et aux recommandations des cours de programmation web.

3.2 La Vue (View)

- **Description :**
  Les vues correspondent aux fichiers de templates (situés dans **/templates**) qui génèrent l'interface utilisateur. Elles utilisent le moteur de templating Jinja2 pour afficher dynamiquement les informations.
- **Fonctionnalités :**
  - **Génération de contenu dynamique :** Les pages (par exemple, la page de profil, la page de connexion ou d'inscription) affichent des données issues des modèles.
  - **Séparation du contenu et de la logique :** Le code HTML est épuré de toute logique métier, ce qui améliore la lisibilité et facilite les modifications visuelles.
- **Avantages :**
  Une interface claire et modulable qui permet de mettre à jour l'apparence de l'application sans impacter la logique métier.

3.3 Le Contrôleur (Controller)

- **Description :**
  Les contrôleurs, regroupés dans le dossier **/controllers**, orchestrent le flux des données entre les modèles et les vues. Ils gèrent les requêtes HTTP, effectuent les vérifications nécessaires et redirigent vers les templates appropriés.

- **Fonctionnalités :**
    - **Gestion des routes :** Par exemple, les routes pour l’authentification, la connexion, l’inscription et la consultation des profils sont définies ici.
    - **Contrôle d’accès et validation :** Les contrôleurs vérifient que l’utilisateur est authentifié avant de permettre l’accès à certaines pages, en s’appuyant sur la gestion des sessions.
- **Avantages :**
  Une logique métier centralisée qui simplifie la maintenance et permet d’assurer la cohérence des règles d’accès à l’application.

3.4 Services

L'application UserVault un service essentiel qui soutiennent l'ensemble de l'architecture backend :

- **Service de Gestion des Utilisateurs :**
  Permet la création, la lecture, la mise à jour et la suppression des informations utilisateur via une interface simplifiée.

4. Fonctionnalités Clés et Sécurité

4.1 Gestion de l’Authentification et des Sessions

- **Authentification :**
  Lorsqu’un utilisateur se connecte, le contrôleur vérifie ses identifiants (en comparant par exemple le mot de passe haché stocké dans la base avec la saisie de l’utilisateur). En cas de succès, l’identifiant utilisateur est stocké dans une session.
- **Gestion des sessions :**
  La session est gérée via un cookie signé à l’aide d’une SECRET_KEY, garantissant l’intégrité et la confidentialité des données. Cette approche permet de :
    - Protéger les données sensibles (les cookies sont marqués HTTPOnly, ce qui limite les risques d’attaques XSS).
    - Permettre une vérification systématique sur chaque requête, redirigeant les utilisateurs non authentifiés vers la page de connexion.
- **Sécurité renforcée :**
  Outre le hachage des mots de passe, la validation des entrées et l’utilisation de méthodes POST pour les opérations sensibles contribuent à sécuriser l’application contre les injections SQL et autres vulnérabilités.

4.2 Interaction avec la Base de Données

- **Utilisation de SQLAlchemy :**
  L’intégration de l’ORM permet de gérer la persistance des données de manière

efficace et sécurisée. Les modèles définis dans **/models** interagissent avec la base de données (souvent PostgreSQL pour un développement local)

- **Création et migration de schémas :**
  Le script d'initialisation (via db.create_all() ou des outils de migration) permet de générer et de mettre à jour la structure de la base de données sans intervention manuelle sur les tables.

5. Conclusion

Le dépôt **UserVault** présente une application web robuste et bien structurée, respectant les principes de l'architecture MVC. Grâce à une organisation claire – avec des dossiers distincts pour les modèles, les vues et les contrôleurs – le projet permet une maintenance facilitée et une extensibilité certaine. La gestion sécurisée des sessions et l'utilisation de SQLAlchemy pour l'interaction avec la base de données démontrent un soin particulier pour la sécurité et la fiabilité du système.

Ce projet constitue une excellente illustration des concepts et des bonnes pratiques enseignées dans le cursus de programmation web en Python. Il démontre qu'en adoptant une approche modulaire et sécurisée, il est possible de développer des applications web évolutives et résilientes face aux défis de la gestion des utilisateurs et de l'authentification.