Faculté des sciences et techniques, Tanger

Système d'Authentification Flask

Rapport Technique

Réaliser par
ZARQI EZZOUBAIR

Rapport technique - 28 mars 2025

Table des matières

1 Introduction

Ce document technique décrit l'implémentation d'un système d'authentification sécurisé utilisant Flask. L'application suit une architecture MVC (Modèle-Vue-Contrôleur) et utilise SQLAlchemy pour la gestion de base de données. Les principales fonctionnalités incluent l'inscription, la connexion, et la gestion de sessions utilisateurs.

2 Structure des Fichiers

```
.
app/
   config.py          # Configuration de l'application
   controllers/       # Contrôleurs (logique métier)
      auth_controller.py  # Gestion authentification
   models/            # Modèles de données
      db.py          # Configuration SQLAlchemy
      user.py        # Modèle utilisateur
   static/            # Fichiers statiques (CSS)
   templates/         # Templates HTML
instance/              # Base de données SQLite
requirements.txt       # Dépendances Python
run.py                 # Point d'entrée de l'application
```

3 Gestion des Sessions

- Stockage de l'ID utilisateur dans un cookie signé
- Mécanisme de sécurité :
  - Clé secrète (SECRET_KEY) pour signer les cookies
  - Cookie HTTPOnly pour prévenir les attaques XSS
  - Session permanente avec expiration configurable
- Workflow :
  1. Authentification réussie → Stockage user_id dans session
  2. Chaque requête vérifie la présence de user_id
  3. Déconnexion → Suppression de user_id de la session

4 Modèle Utilisateur

- Fichier : `app/models/user.py`
- Structure :

    - `id` : Identifiant unique (clé primaire)
    - `username` : Nom d'utilisateur unique
    - `email` : Adresse email unique
    - `password_hash` : Mot de passe haché
- Méthodes :
    - `set_password(password)` : Hachage du mot de passe
    - `check_password(password)` : Vérification du hash

5 Implémentation dans Flask

5.1 Configuration Initiale

- Initialisation SQLAlchemy dans `db.py`
- Création des tables avec `db.create_all()`
- Intégration du modèle User dans les contrôleurs

5.2 Protection des Routes

Exemple de code pour protéger une route :

```
@app.route('/profile')
def profile():
    if 'user_id' not in session:
        flash('Connectez-vous pour accéder à cette page')
        return redirect(url_for('auth.login'))
    user = User.query.get(session['user_id'])
    return render_template('profile.html', user=user)
```

6 Amélioration du Système de Gestion de Tâches

6.1 Description Fonctionnelle

L'implémentation de la gestion des tâches ajoute les fonctionnalités suivantes :

- Création de tâches avec titre et description
- Marquage instantané des tâches comme terminées
- suppression sécurisée des tâches
- Affichage visuel différencié pour les tâches complétées
- Formulaire intégré pour une création rapide

6.2 Implémentation Technique

6.2.1 Modèle de Données

Structure de la table `Task` :

```
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text)
    completed = db.Column(db.Boolean, default=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

6.2.2 Contrôleurs

Routes principales ajoutées :
— `/tasks` : Liste des tâches avec formulaire intégré
— `/tasks/<id>/delete` : Suppression sécurisée

6.2.3 Sécurité

Mesures implémentées :
— Vérification systématique de l'appartenance des tâches
— Protection CSRF via méthodes POST
— Validation des entrées utilisateur

6.3 Interface Utilisateur

Modifications apportées au template :
— Formulaire inline pour création rapide
— Boutons d'action contextuels
— Feedback visuel instantané

6.4 Avantages

Cette implémentation offre :
— Une interaction utilisateur fluidifiée
— Un traitement instantané des actions
— Une sécurité renforcée
— Une expérience utilisateur cohérente
— Une base extensible pour de futures fonctionnalités

7 Conclusion

Ce système d'authentification moderne représente une implémentation robuste des bonnes pratiques de développement web sécurisé. Fondé sur l'architecture MVC, il démontre une séparation claire des responsabilités entre la gestion des données (modèles), la logique métier (contrôleurs) et l'interface utilisateur (vues), offrant ainsi une base extensible pour des fonctionnalités futures.

L'utilisation combinée de Flask et SQLAlchemy permet une intégration transparente avec divers systèmes de stockage de données, tandis que le mécanisme de sessions sécurisées garantit une gestion fiable des états utilisateurs. La mise en œuvre du hachage de mots de passe via Werkzeug Security élimine les risques liés au stockage de credentials en clair, renforçant la conformité RGPD.

Les choix architecturaux remarquables incluent :

— Une abstraction efficace de la couche données via l'ORM SQLAlchemy
— Une gestion centralisée de la configuration pour différents environnements
— Des templates HTML modulaires favorisant la réutilisation du code
— Un système de routage hiérarchique via les Blueprints Flask

Par son équilibre entre simplicité d'implémentation et rigueur sécuritaire, cette solution démontre qu'une infrastructure d'authentification fiable peut être développée avec des technologies open-source modernes, tout en restant adaptable aux exigences changeantes des applications web contemporaines.