



Rapport project

Python

for Othman bakkali
by mohamed said boufanzi

1. Introduction

Ce rapport présente une analyse détaillée du système de gestion d'utilisateurs développé avec Flask. L'application permet aux utilisateurs de créer un compte, se connecter, et gérer leur profil utilisateur. Ce document explique les technologies utilisées, l'architecture du code, et les mécanismes d'authentification et de sécurité implémentés.

2. Technologies Utilisées

2.1 Framework et Langage

- **Python**: Langage de programmation principal
- **Flask**: Framework web léger pour Python
- **Jinja2**: Moteur de templates intégré à Flask

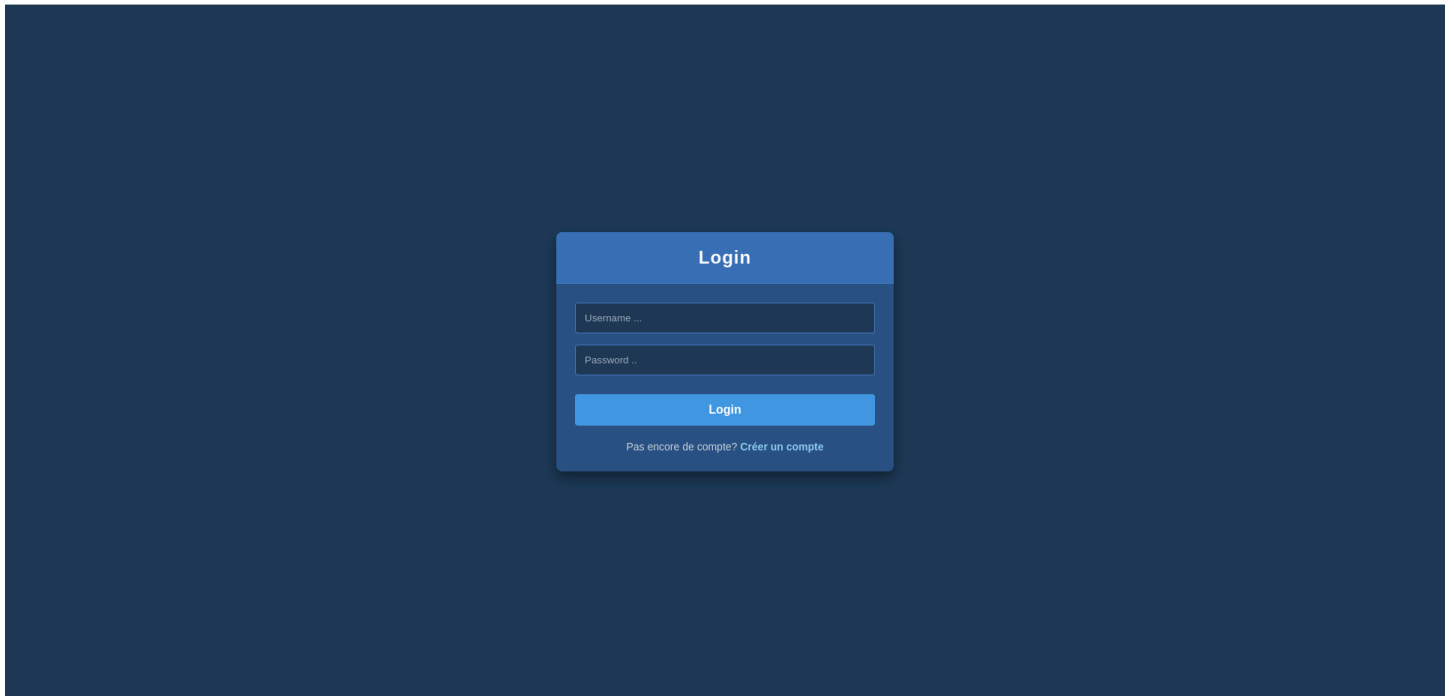
2.2 Base de Données

- **MySQL**: Système de gestion de base de données relationnelle
- **SQLAlchemy**: ORM (Object-Relational Mapping) pour Python
- **Flask-SQLAlchemy**: Extension Flask pour intégrer SQLAlchemy

2.3 Sécurité et Authentification

- **Werkzeug**: Bibliothèque pour le hachage des mots de passe
- **Sessions Flask**: Mécanisme de gestion des sessions utilisateur
- **python-dotenv**: Gestion des variables d'environnement

3. login page (home)



3.1 Fonctionnalités

La page de connexion sert de point d'entrée principal à l'application et offre les fonctionnalités suivantes:

- Formulaire de connexion avec champs pour nom d'utilisateur et mot de passe
- Validation des identifiants utilisateur
- Affichage des messages d'erreur en cas d'échec de connexion
- Lien vers la page d'inscription pour les nouveaux utilisateurs

3.2 Implémentation Technique

3.2.1 Route Flask

```
```python
@auth_bp.route('/login', methods=['POST', 'GET'])
def login():
```

```

if request.method == 'POST':
 username = request.form.get('username')
 password = request.form.get('password')

 # Recherche de l'utilisateur dans la base de données
 user = User.query.filter_by(username=username).first()

 # Vérification du mot de passe avec Werkzeug
 if user and check_password_hash(user.hashed_password, password):
 # Création de la session utilisateur
 session['user_id'] = user.id
 session['username'] = user.username
 session['email'] = user.email
 flash('Connexion réussie!', 'success')
 return redirect(url_for('user_bp.dashboard'))

 flash('Nom d\'utilisateur ou mot de passe invalide', 'danger')
 return redirect(url_for('auth_bp.login'))

return render_template('login.html')

```

### 3.2.2 Gestion des Sessions

La page de connexion initialise une session Flask lorsque l'authentification réussit. Cette session contient:

- L'identifiant de l'utilisateur (`user\_id`)
- Le nom d'utilisateur (`username`)
- L'adresse email (`email`)

Ces informations sont stockées côté serveur, tandis qu'un cookie de session est envoyé au navigateur client. Ce cookie est crypté à l'aide de la clé secrète définie dans la configuration de l'application.

### 3.2.3 Sécurité

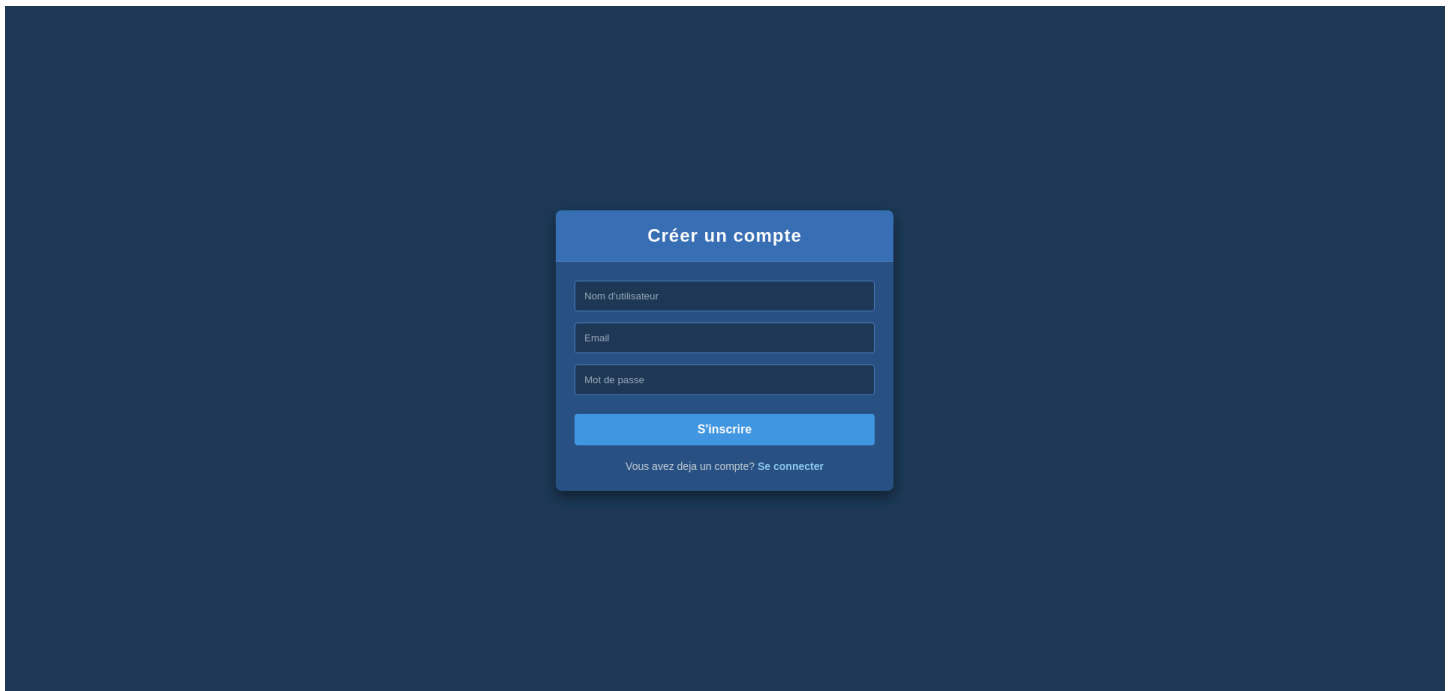
Plusieurs mesures de sécurité sont implémentées:

- Les mots de passe ne sont jamais stockés en clair, mais hachés avec Werkzeug
- La vérification du mot de passe utilise une comparaison sécurisée pour éviter les attaques temporelles
- Les messages d'erreur sont génériques pour ne pas révéler si un nom d'utilisateur existe
- La session est cryptée avec une clé secrète stockée dans les variables d'environnement

### 3.2.4 Interface Utilisateur

L'interface utilisateur est implémentée en HTML/CSS avec un design responsive. Les messages flash de Flask sont utilisés pour afficher les notifications à l'utilisateur.

## 4. sign up page



Créer un compte

Nom d'utilisateur

Email

Mot de passe

S'inscrire

Vous avez déjà un compte? [Se connecter](#)

### 4.1 Fonctionnalités

La page d'inscription permet aux nouveaux utilisateurs de créer un compte avec les fonctionnalités

suivantes:

- Formulaire d'inscription avec champs pour nom d'utilisateur, email et mot de passe
- Validation des données saisies
- Vérification de l'unicité du nom d'utilisateur et de l'email
- Création automatique du compte et connexion immédiate

## 4.2 Implémentation Technique

### 4.2.1 Route Flask

```
```python
@auth_bp.route('/signup', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')

        # Vérification de l'unicité de l'email
        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            return render_template('signup.html', message={'error': 'Cet email est déjà utilisé'})

        # Hachage du mot de passe pour la sécurité
        hashed_password = generate_password_hash(password)

        # Création d'un nouvel utilisateur
        new_user = User(username=username, email=email, hashed_password=hashed_password)
        db.session.add(new_user)
        db.session.commit()

        # Initialisation de la session utilisateur
        session['user_id'] = new_user.id
        session['username'] = new_user.username
```

```
session['email'] = new_user.email
```

```
flash('Compte créé avec succès!', 'success')  
return redirect(url_for('user_bp.dashboard'))
```

```
return render_template('signup.html')
```

4.2.2 Sécurisation des Mots de Passe

Le processus de hachage des mots de passe est une étape cruciale:

1. Le mot de passe en clair est reçu du formulaire
2. La fonction `generate_password_hash()` de Werkzeug crée un hash sécurisé
3. Seul le hash est stocké dans la base de données
4. Le mot de passe original n'est jamais stocké ni conservé

Werkzeug utilise par défaut l'algorithme PBKDF2 avec un sel aléatoire, ce qui protège contre:

- Les attaques par dictionnaire
- Les attaques par force brute
- Les attaques par table arc-en-ciel (rainbow table)

4.2.3 Interaction avec la Base de Données

SQLAlchemy gère l'interaction avec la base de données MySQL:

1. Vérification de l'unicité de l'email avec une requête de type `filter_by`
2. Création d'une nouvelle instance du modèle `User`
3. Ajout de l'instance à la session de base de données avec `db.session.add()`
4. Validation des changements avec `db.session.commit()`

4.2.4 Initialisation de Session

Après l'inscription réussie, une session est immédiatement créée pour l'utilisateur, évitant ainsi la nécessité de se connecter séparément.

Créer un compte

helloworld

boufanzhi.mohamedsaid@etu.uae.ac.ma

.....

S'inscrire

Vous avez deja un compte? [Se connecter](#)

5. manage user page

Tableau de bord

Déconnexion

Gestion du Profil

ID	Nom d'utilisateur	Email	Actions
3	helloworld	boufanzhi.mohamedsaid@etu.uae.ac.ma	Modifier

5.1 Fonctionnalités

La page de gestion utilisateur permet aux utilisateurs connectés de:

- Visualiser leurs informations personnelles

- Modifier leur nom d'utilisateur et adresse email
- Se déconnecter de l'application

5.2 Implémentation Technique

5.2.1 Route de Tableau de Bord

```
```python
@user_bp.route("/dashboard", methods=['GET'])
def dashboard():
 # Vérification de l'authentification
 if 'user_id' not in session:
 flash("Vous devez vous connecter d'abord.", "warning")
 return redirect(url_for('auth_bp.login'))

 # Préparation des données utilisateur
 user_info = {
 "id": session.get("user_id"),
 "username": session.get("username"),
 "email": session.get("email"),
 }

 return render_template('manage_user.html', user=user_info)
```
```

Route de Modification Utilisateur

```
```python
@user_bp.route('/edit_user', methods=['POST'])
def edit_user():
 # Vérification de l'authentification
 if 'user_id' not in session:
 flash("Vous devez vous connecter d'abord.", "warning")
 return redirect(url_for('auth_bp.login'))
```

```

Récupération des données du formulaire
user_id = session.get('user_id')
username = request.form.get('username')
email = request.form.get('email')

Mise à jour des informations utilisateur
user = User.query.get(user_id)
if user:
 user.username = username
 user.email = email
 db.session.commit()

Mise à jour de la session
session['username'] = username
session['email'] = email
flash('Informations mises à jour avec succès!', 'success')
else:
 flash("Utilisateur non trouvé!", "danger")

return redirect(url_for('user_bp.dashboard'))
...

```

### 5.2.3 Mécanisme d'Autorisation

Cette page implémente un mécanisme d'autorisation basé sur les sessions:

1. Vérification de la présence de `user\_id` dans la session
2. Redirection vers la page de connexion si l'utilisateur n'est pas authentifié
3. Accès aux fonctionnalités uniquement pour les utilisateurs autorisés

Ce mécanisme garantit que seuls les utilisateurs connectés peuvent accéder à leurs informations personnelles et les modifier.

### 5.2.4 Mise à Jour de Session

Lors de la modification des informations utilisateur, la session est également mise à jour pour refléter les changements:

```
```python
session['username'] = username
session['email'] = email
```
```

Cette synchronisation entre la base de données et la session garantit la cohérence des données affichées à l'utilisateur.

### #### 5.2.5 Interface Utilisateur Interactive

L'interface utilisateur utilise JavaScript pour:

- Afficher un modal d'édition lorsque l'utilisateur clique sur le bouton "Modifier"
- Pré-remplir le formulaire avec les informations actuelles de l'utilisateur
- Soumettre les modifications via une requête POST

## 6. Conclusion

Ce système de gestion d'utilisateurs démontre l'implémentation de concepts clés en matière de sécurité web:

- Authentification sécurisée avec hachage des mots de passe
- Autorisation basée sur les sessions
- Protection des données utilisateur
- Interface utilisateur intuitive et responsive

L'architecture modulaire de l'application permet une maintenance facile et des extensions futures, comme l'ajout de fonctionnalités d'authentification avancées ou l'intégration avec d'autres services.