

Université Abdelmalek Essaadi

Faculté des Sciences et Techniques de Tanger

# **-Projet Python-Flask- Authentification et Gestion de Finances**

Encadré Par : Pr.Bakkali Othman

Réaliser Par : Zeroual Hiba

## Introduction:

Ce projet, développé avec **Python**, **Flask** et **MySQLAlchemy**, permet aux utilisateurs de s'inscrire, se connecter et suivre leurs finances via un **dashboard interactif**. Ils peuvent ajouter leurs **revenus et dépenses**, visualiser l'évolution de leurs finances avec un graphique via **Chart.js** et sauvegarder leurs données en **base de données MySQL**. L'application combine ainsi gestion financière et visualisation de données.

### → Fonctionnalités :

- **Inscription et Connexion**

Les utilisateurs peuvent s'inscrire avec un nom, un e-mail et un mot de passe. Lors de l'inscription, le mot de passe est **crypté** avant d'être stocké en base de données. Une fois inscrits, les utilisateurs peuvent se connecter à l'application avec leur e-mail et mot de passe.

- **Dashboard Financier**

Après la connexion, les utilisateurs sont redirigés vers un **dashboard** qui affiche leurs transactions (revenus et dépenses) et leur **solde total** (revenus - dépenses) et un diagramme de RD. Un utilisateur peut visualiser et ajouter de nouvelles transactions.

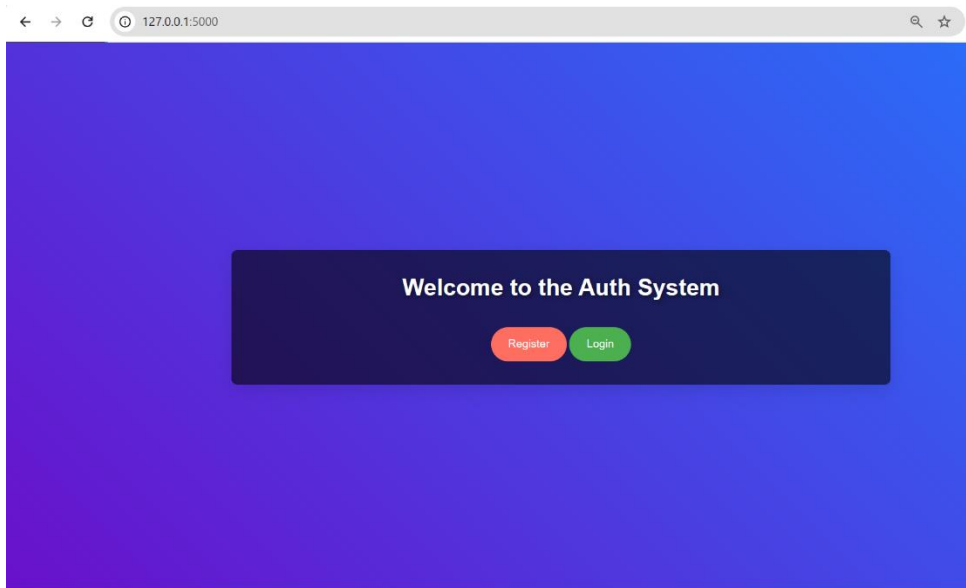
- **Ajout et Suppression de Transactions**

Les utilisateurs peuvent ajouter des transactions sous forme de **revenus** ou **dépenses** avec un montant et une description. Ces transactions sont sauvegardées dans une base de données **MySQL** grâce à **SQLAlchemy**. Ils peuvent également supprimer des transactions existantes.

- **Base de Données et Sécurité**

La base de données est gérée par **SQLAlchemy** et utilise **MySQL** pour stocker les informations des utilisateurs et des transactions. Le mot de passe des utilisateurs est sécurisé avec **bcrypt**.

→ **Page Index** : une page d'accueil simple avec HTML et Bootstrap pour un système d'authentification. Le fond est un dégradé dynamique, et le contenu est centré grâce à Flexbox.



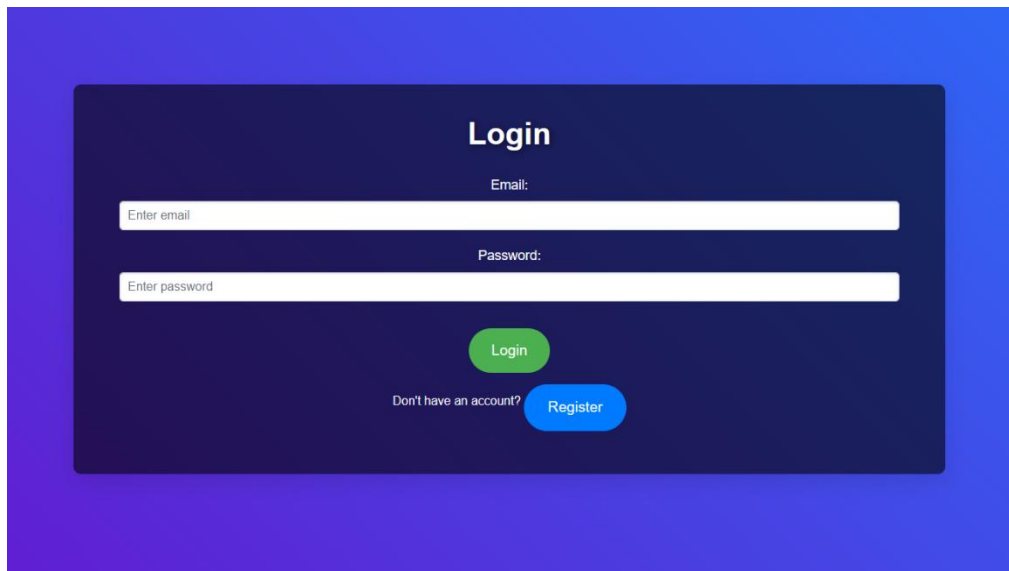
→ J'ai utilisé **Bootstrap 4** pour la mise en page et les boutons ,J'ai ajouté deux boutons : un pour l'inscription et un pour la connexion, avec des liens vers les pages appropriées.

➤ Puis l'utilisateur clique sur Register :

→ La page d'inscription offre aux utilisateurs une interface claire et intuitive pour créer un compte. Grâce à **Bootstrap**, le formulaire est structuré de manière responsive, avec des champs bien espacés et une mise en page soignée.

→ La route **/register** permet aux utilisateurs de s'inscrire en envoyant leurs informations via un formulaire. Lorsqu'une requête **POST** est reçue, les données sont récupérées, un nouvel utilisateur est créé et enregistré dans la base de données. Une fois l'inscription réussie, l'utilisateur est redirigé vers la page de connexion.

→ **Page login** : est la porte d'entrée vers le dashboard.



- **Méthode POST** : Envoie les données de manière sécurisée (pas d'exposition dans l'URL)

### Interactions avec la BDD (SQLAlchemy)

- **Vérification de l'utilisateur** : Requête pour trouver l'email en base (ex: `User.query.filter_by(email=email).first()`).
  - **Hachage BCrypt** : Vérification du mot de passe avec `bcrypt.check_password_hash(user.password, password)`.
- Si valide : une session Utilisateur est créer `session['email']`.  
Si invalide : Affiche un message d'erreur.
- Une fois la connexion est réussie : **return redirect('/dashboard')**
  - La **route /dashboard** vérifie la session avant d'afficher les donnée  
`@app.route('/dashboard')`  
`def dashboard():`  
    if 'email' not in session:  
        return redirect('/login') # Accès refusé si non connecté



→ **Page Dashboard** : une interface intuitive pour la gestion des finances personnelles, combinant visualisation des données, interactivité et facilité d'utilisation.

- **Structure et style** : Utilisation de Bootstrap pour la mise en page et de Chart.js pour le graphique, avec un fond dégradé et un design moderne, incluant des ombres pour un effet visuel attrayant.

- **Affichage des informations utilisateur** : Le nom, l'email de l'utilisateur connecté, et le solde total sont affichés avec un badge coloré.

- **Formulaire d'ajout de transactions** : Permet à l'utilisateur d'ajouter des revenus ou des dépenses avec description, montant et type.

- **Tableau des transactions** : Affiche les transactions sous forme de tableau avec un bouton "Supprimer" pour chaque transaction.

- **Graphique interactif** : Utilisation de Chart.js pour un graphique montrant l'évolution mensuelle des revenus et dépenses, avec des couleurs distinctes pour chaque type.
- **Déconnexion** : Un bouton pour se déconnecter et retourner à la page de connexion.

- **Calculs financiers :**

- **Solde total** : Différence entre revenus et dépenses (sum() sur les transactions).
- Transmet ces données au template (render\_template).

---

➤ **.add\_transaction() (Route /add\_transaction)**

**Rôle** : Ajoute une nouvelle transaction (revenu/dépense).

**Fonctionnalités** :

- **Validation de session :**
    - Bloque l'accès si non connecté (même mécanisme que /dashboard).
  - **Traitement du formulaire :**
    - Récupère le type (revenu/dépense), le montant, et la description.
    - Crée un objet Transaction et l'enregistre en BDD (**db.session.add()**).
  - **Feedback utilisateur :**
    - Affiche un message de succès (**flash()**).
    - Redirige vers /dashboard pour actualiser l'affichage.
- 

### **.delete\_transaction() (Route /delete\_transaction/<id>)**

**Rôle :** Supprime une transaction existante.

**Fonctionnalités :**

- **Protection :** Vérifie la session et l'existence de la transaction.
  - **Action :**
    - Supprime la transaction via son ID (**Transaction.query.get(id)**).
    - Met à jour la BDD (**db.session.commit()**).
  - **Feedback :**
    - Message de confirmation et redirection vers /dashboard.
- 

## **2. Interactions avec la Base de Données**

### **2.1. Modèle User**

- **Champs :** id, name, email, password (haché).
- **Méthode critique :**
  - **check\_password()** : Compare le mot de passe saisi avec le hachage stocké (via bcrypt).

### **2.2. Modèle Transaction**

- **Champs :**
  - user\_id (clé étrangère), type, amount, description, date.
- **Lien avec l'utilisateur :**

- Permet de filtrer les transactions par utilisateur (**user\_id=user.id**).
- 

### 3. Sécurité et Gestion des Erreurs

- **Sessions Flask :**

- Stockage de session['email'] pour maintenir la connexion.
- Suppression lors de la déconnexion (**session.pop('email')**).

- **Validation côté serveur :**

- Toutes les routes vérifient la session avant toute action.
  - Les transactions sont liées à l'utilisateur connecté (pas de suppression/modification par un autre).
- 

### 4. Visualisation des Données (Frontend)

- **Graphique Chart.js :**

- Affiche des données statiques (pour l'instant).

- **Tableau des transactions :**

- **Boucle Jinja2** (**{% for transaction in transactions %}**) pour lister chaque entrée.
- **Badges colorés** (badge-success pour les revenus, badge-danger pour les dépenses).

## **Conclusion :**

Ce projet de gestion financière personnelle, développé avec **Python, Flask** et **SQLAlchemy**, illustre parfaitement comment une application web peut allier fonctionnalités pratiques et technologies modernes pour répondre à un besoin concret. Grâce à une architecture bien pensée et une interface intuitive, les utilisateurs peuvent suivre leurs finances en temps réel, ajouter ou supprimer des transactions, et visualiser leurs données via un dashboard interactif.

### ❖ **Points Forts du Projet-**

#### → **Sécurité Renforcée :**

Hachage des mots de passe avec bcrypt pour protéger les données sensibles.

Gestion des sessions pour un accès sécurisé au dashboard.

#### → **Expérience Utilisateur Optimisée :**

Interface conviviale (Bootstrap) avec un design moderne (dégradés, effets visuels).

Feedback immédiat (messages flash, calculs dynamiques du solde).

#### → **Fonctionnalités Clés :**

Inscription/Connexion simplifiée.

Gestion complète des transactions (ajout/suppression).

Visualisation des données via un graphique Chart.js (revenus vs dépenses).

#### → **Base de Données Robustes :**

Modèles SQLAlchemy bien structurés (User, Transaction).

Relations claires (clé étrangère user\_id pour lier les transactions aux utilisateurs).

## **Bilan**

Ce projet démontre avec succès comment intégrer des concepts techniques avancés (sécurité, bases de données, visualisation) dans une application accessible et utile au quotidien. Il sert de base solide pour des extensions plus complexes, tout en respectant les bonnes pratiques de développement web.

- **Ressources :**

- [Flask](#) : Pour maîtriser les routes, sessions et templates.

- [SQLAlchemy](#) : Modélisation avancée des bases de données.

- [Chart.js](#) : Créer des visualisations dynamiques .

- [CSS Gradients](#) : Personnaliser les dégradés arrière-plan.



