



# **RAPPORT DE PROJET** APPLICATION WEB LOGIN / LOGOUT

**2024 -2025**

---

**Encadré par**  
Pr. Othman Bakkali

**Élaboré par**  
CHENAOUI Hiba

# — Introduction

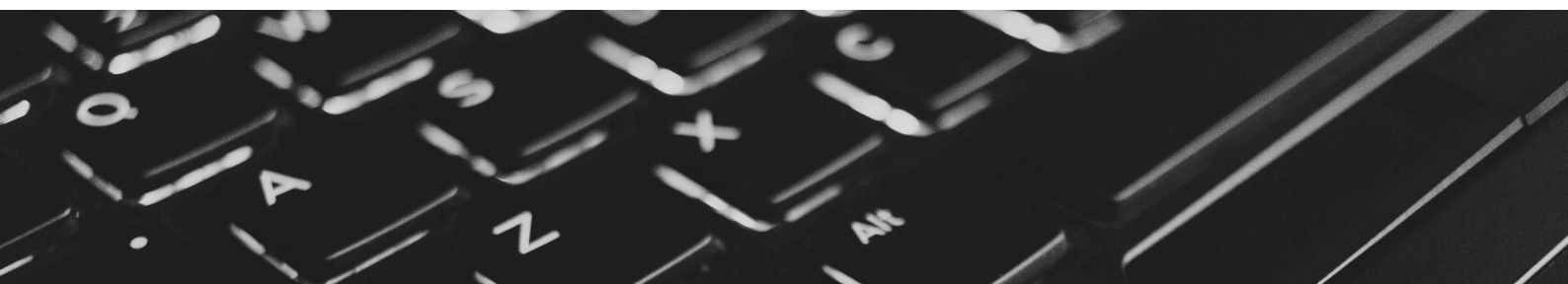
L'authentification des utilisateurs est une fonctionnalité essentielle dans le développement d'applications web modernes. Elle permet de garantir la sécurité des données et des services en ligne en s'assurant que seules les personnes autorisées puissent accéder à certaines informations ou effectuer des actions spécifiques.

Le projet "Application Web de Login/Logout avec Python et Flask" vise à créer une application simple mais sécurisée permettant aux utilisateurs de s'inscrire, se connecter, et se déconnecter. L'application utilise Flask, un framework léger et flexible pour le développement web en Python, et SQLite, une base de données relationnelle légère, pour stocker les informations des utilisateurs.

Ce projet aborde les aspects de base de la gestion des utilisateurs, notamment :

- L'inscription : Permet aux nouveaux utilisateurs de créer un compte avec un mot de passe sécurisé.
- La connexion : Offre une interface permettant aux utilisateurs d'accéder à leur compte en vérifiant leurs identifiants.
- La déconnexion : Permet aux utilisateurs de se déconnecter en toute sécurité.

L'objectif de ce projet est de mettre en œuvre des compétences en gestion des sessions utilisateur, en sécurisation des mots de passe avec la bibliothèque Werkzeug, et en manipulation d'une base de données SQLite pour stocker et gérer les informations des utilisateurs.



# Technologies utilisées



## Flask

Flask est un framework web Python léger et flexible, idéal pour des applications simples. Il gère les routes, la logique de l'application et la gestion des sessions utilisateur. Dans ce projet, Flask permet de définir les différentes pages (connexion, inscription, etc.) et de gérer les interactions avec l'utilisateur via des requêtes HTTP.

## SQLite

SQLite est une base de données légère, intégrée et sans serveur. Elle est utilisée pour stocker les informations des utilisateurs (nom, email, mot de passe). Sa simplicité et sa facilité d'intégration font d'elle un choix idéal pour ce projet de petite envergure.

## Werkzeug

Werkzeug est une bibliothèque Python utilisée pour sécuriser l'application. Elle permet de hacher les mots de passe des utilisateurs avant de les stocker dans la base de données, garantissant ainsi leur sécurité. Elle fournit également des outils pour gérer les sessions utilisateurs de manière sécurisée.

# — Architecture du projet

L'architecture du projet repose sur le modèle MVC, qui est un modèle de conception largement utilisé dans le développement d'applications web. Cette architecture permet de séparer les différentes préoccupations de l'application, facilitant ainsi sa maintenance, son évolutivité et sa gestion.

## Modèle

Le Modèle représente la logique de gestion des données. Dans le cadre de ce projet, il est responsable de la gestion des utilisateurs, y compris la création, la récupération et la mise à jour des informations des utilisateurs.

### Gestion des utilisateurs

---

Le modèle se charge de l'insertion des nouvelles données des utilisateurs dans la base de données SQLite, comme les informations personnelles (nom, prénom, email, etc.) et les informations de sécurité (mot de passe haché).

### Sécurisation des mots de passe

---

Le modèle utilise la bibliothèque Werkzeug pour hacher les mots de passe avant de les stocker dans la base de données, garantissant ainsi la sécurité des données sensibles. Lors de la connexion, le modèle vérifie le mot de passe.

### Interaction avec la base de données

---

Le modèle interagit directement avec la base de données SQLite pour récupérer les informations des utilisateurs lors de la connexion et insérer de nouvelles données lors de l'inscription.

# Vue

La Vue représente la présentation de l'application, c'est-à-dire l'interface utilisateur. Dans ce projet, la vue est constituée des fichiers HTML qui sont utilisés pour afficher les différentes pages de l'application.

## Pages HTML

---

Les pages HTML comprennent des formulaires pour l'inscription, la connexion, et un tableau de bord pour l'utilisateur une fois connecté. Ces pages sont rendues dynamiquement par Flask, en utilisant les données fournies par le contrôleur et le modèle.

## L'interface

---

Les fichiers HTML sont stockés dans le dossier templates/, ce qui permet de séparer la logique de présentation du reste de l'application. Chaque vue fournit une expérience utilisateur simple et claire, en permettant à l'utilisateur d'interagir avec l'application.

# Contrôleur

## Gestion des requêtes

---

Le contrôleur définit les routes correspondant aux différentes actions de l'application. Lorsqu'une requête est reçue, le contrôleur exécute la logique nécessaire pour traiter les données. Il récupère les informations envoyées par l'utilisateur, les transmet au modèle pour les ajouter à la base de données, puis redirige l'utilisateur vers la page de connexion une fois l'opération terminée.

## Gestion des sessions

---

Le contrôleur est également responsable de la gestion des sessions. Lorsqu'un utilisateur se connecte avec succès, le contrôleur crée une session pour maintenir l'état de l'utilisateur, permettant ainsi d'afficher du contenu personnalisé. Lors de la déconnexion, le contrôleur efface la session de l'utilisateur pour le déconnecter proprement de l'application.

# Conclusion

En résumé, l'architecture MVC dans ce projet permet de structurer le code de manière claire et logique. Le modèle s'occupe de la gestion des données et de la sécurité, la vue prend en charge l'interface utilisateur, et le contrôleur gère les requêtes et la logique de traitement des données. Cette séparation des responsabilités facilite la maintenance du code et permet de développer chaque composant de manière indépendante tout en garantissant une interaction fluide entre eux.

# Explication du code et structure des fichiers

## Fichier run.py

Le fichier run.py sert à démarrer notre application. Il utilise une fonction appelée `create_app()` qui prépare l'application avec tout ce dont elle a besoin (comme les routes et la configuration). Si on lance ce fichier, il démarre le site en mode développeur, ce qui permet de voir les erreurs facilement et de tester plus rapidement.

## Fichier APP/\_\_init\_\_.py

Le fichier `__init__.py` sert à initialiser l'application Flask. Il commence par importer le framework Flask, puis les contrôleurs user et auth ainsi que leurs blueprints, afin qu'ils soient reconnus et utilisés par l'application. Il importe également la configuration (Config), qui contient des éléments essentiels comme la `secret_key` et les paramètres de connexion à la base de données. Enfin, la fonction `create_app()` crée une instance de l'application Flask, y applique la configuration, enregistre les blueprints, et renvoie l'application prête à être exécutée.

## Fichier user\_controller.py

Ce fichier gère la route `/`, qui correspond à la page d'accueil personnalisée d'un utilisateur connecté. Il utilise un Blueprint pour organiser les routes liées à l'utilisateur. Lorsqu'un utilisateur accède à cette route, la logique vérifie s'il est connecté via la session (grâce à la présence de `user_id` dans la session). Si c'est le cas, le contrôleur affiche la vue appropriée avec un message personnalisé. Sinon, il redirige automatiquement l'utilisateur vers la page de connexion (`/login`). Ce mécanisme permet de protéger l'accès à la page et d'assurer une navigation fluide selon l'état de connexion de l'utilisateur.

## Fichier auth\_controller.py

Le fichier `auth_controller.py` contient les routes et la logique de gestion de l'authentification des utilisateurs, notamment l'inscription, la connexion et la déconnexion. Il utilise un Blueprint nommé `auth_bp` qui est ensuite enregistré dans l'application Flask. Voici une explication détaillée de son fonctionnement :

### Route /signup

La route `/signup`, accessible via les méthodes HTTP GET et POST, permet à un utilisateur de s'inscrire. Lorsqu'un utilisateur accède à la route avec la méthode GET, il est présenté avec le formulaire d'inscription. Si la méthode est POST, cela signifie que l'utilisateur a soumis le formulaire avec ses informations (prénom, nom, email, mot de passe). Avant de créer le nouvel utilisateur, le contrôleur vérifie si l'email est déjà enregistré dans la base de données. Si c'est le cas, un message d'erreur s'affiche pour prévenir l'utilisateur que l'email est déjà utilisé. Si l'email est unique, un nouvel utilisateur est créé en utilisant le modèle `User` et les informations reçues, puis l'utilisateur est redirigé vers la page de connexion.

### Route /login

La route `/login`, également disponible via GET et POST, permet à un utilisateur d'entrer ses identifiants pour se connecter. Lorsqu'un utilisateur soumet ses informations via POST, le contrôleur récupère l'email et le mot de passe fournis, puis utilise le modèle `User` pour rechercher un utilisateur avec l'email donné. Si l'utilisateur existe et que le mot de passe fourni correspond à celui stocké (vérification effectuée via la méthode `check_password`), les informations de l'utilisateur (ID et prénom) sont stockées dans la session. Cela permet de maintenir l'état de la connexion et d'afficher des informations personnalisées sur la page d'accueil. Si les identifiants sont incorrects, un message d'erreur est affiché.

### Route /logout

la route `/logout` permet de déconnecter un utilisateur. Lorsque l'utilisateur accède à cette route, la session est complètement effacée à l'aide de la méthode `session.clear()`, et l'utilisateur est redirigé vers la page de connexion, ce qui lui permet de se reconnecter avec de nouveaux identifiants si nécessaire.



## Fichier config.py

Le fichier config.py contient les paramètres de configuration de l'application Flask. Il définit une SECRET\_KEY utilisée pour sécuriser les sessions et autres opérations cryptographiques, ainsi que le chemin vers la base de données SQLite (app.db) qui stocke les données des utilisateurs. Ce fichier permet de centraliser la configuration et de la réutiliser facilement dans l'application via app.config.from\_object(Config).

## Dossier templates

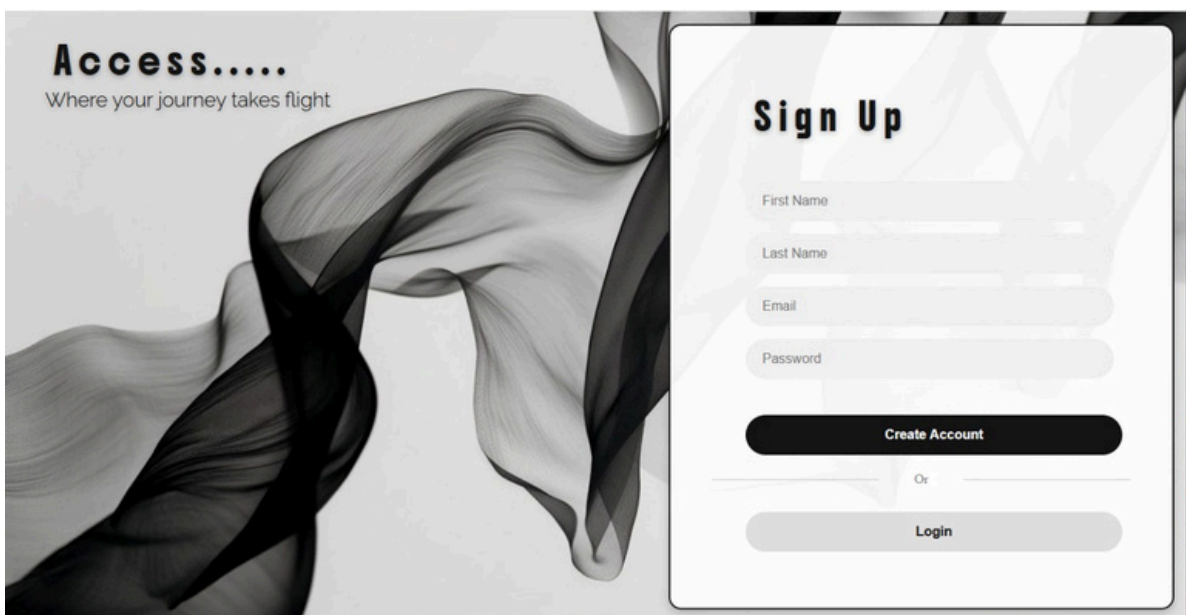
Le dossier templates/ contient les fichiers HTML pour afficher les pages du site. Flask utilise Jinja2 pour intégrer des variables Python dans ces fichiers.

- signup.html : Formulaire d'inscription pour les nouveaux utilisateurs.
- login.html : Formulaire de connexion pour les utilisateurs existants.
- home.html : Page d'accueil affichée après la connexion, qui montre un message de bienvenue.

---

### la page d'inscription

---



**Access.....**  
Where your journey takes flight

**Sign Up**

First Name

Last Name

Email

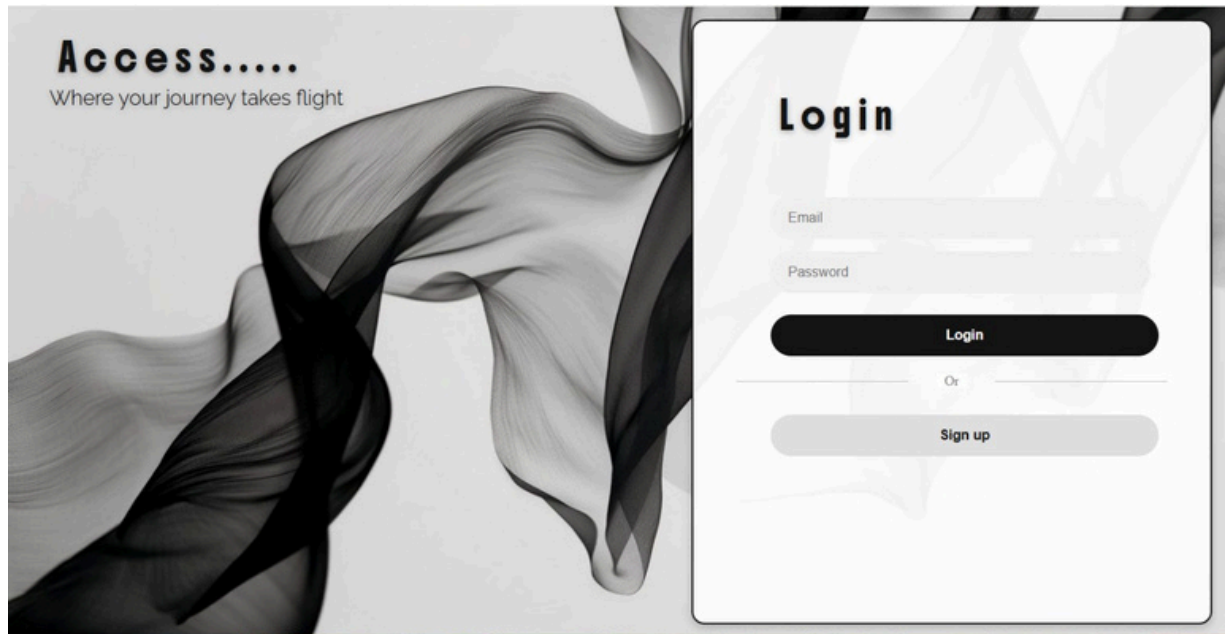
Password

**Create Account**

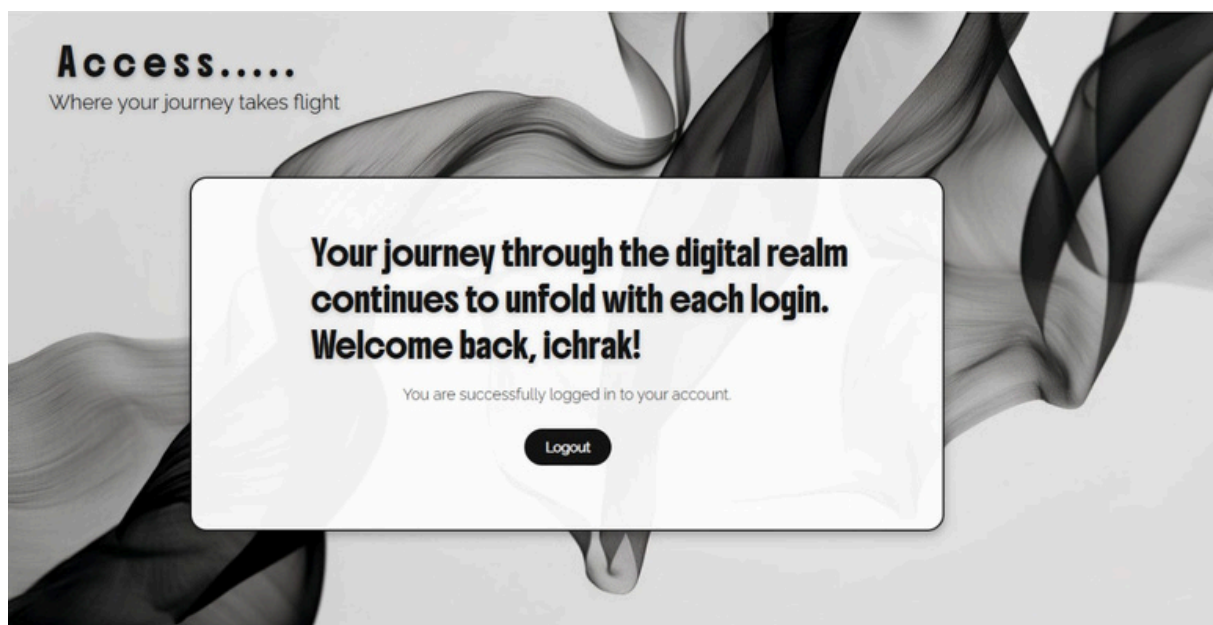
Or

**Login**

## la page de connexion



## la page d'accueil





# — Conclusion

Ce projet a permis de mettre en œuvre une application web simple mais fonctionnelle en utilisant Flask, Python, et SQLite pour gérer l'authentification des utilisateurs. À travers la mise en place des fonctionnalités de connexion, d'inscription et de déconnexion, ce projet a permis de démontrer une gestion sécurisée des utilisateurs en utilisant des sessions et un stockage sécurisé des mots de passe avec la bibliothèque Werkzeug.

Le modèle MVC a été appliqué pour organiser le code de manière structurée, en séparant la logique métier (modèle), la gestion des routes (contrôleur) et la présentation des données (vue). Cette approche a non seulement facilité la gestion et l'évolution du code, mais a aussi renforcé la lisibilité et la maintenabilité de l'application.

En conclusion, ce projet offre une bonne base pour des applications web sécurisées et extensibles, et met en lumière l'importance d'une bonne organisation du code dans le cadre de la programmation web. Il fournit également une première approche de la gestion des utilisateurs et de la sécurité des données dans une application web.