

Université Abdelmalek Essaadi

Faculté des Sciences et Techniques de Tanger

Rapport de Mini-Projet:

Système de Gestion d'Articles avec Authentification

Encadré Par : Pr.Bakkali Othman

Réaliser Par : EL MEHDI EL KHALDI

- **Introduction**

Ce rapport présente un mini-projet web développé avec Python, Flask et SQLAlchemy. L'application, nommée "MonApp", permet aux utilisateurs de s'authentifier, de gérer leur profil, et de créer, visualiser et gérer des articles.

Fonctionnalités Principales

1. Système d'Authentification

- Inscription (création de compte)
- Connexion sécurisée
- Déconnexion
- Récupération de mot de passe

2. Gestion de Profil

- Affichage des informations utilisateur
- Modification des informations (nom, email)

3. Gestion d'Articles

- Création d'articles avec titre, contenu et image
- Affichage de la liste des articles
- Visualisation d'un article individuel
- Modification et suppression d'articles

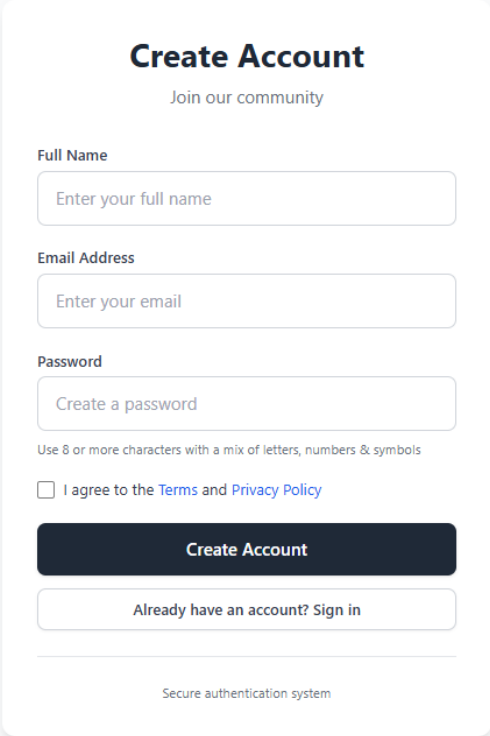
Les pages :

○ **Création de Compte (Register)**

L'interface d'inscription présente un formulaire élégant conçu avec Tailwind CSS, offrant une expérience utilisateur fluide et responsive. Le formulaire comprend:

- Champ pour le nom complet avec validation en temps réel
- Champ pour l'adresse email avec vérification du format
- Champ pour le mot de passe avec indicateur de force
- Case à cocher pour l'acceptation des conditions d'utilisation
- Bouton de création de compte avec état de chargement

En arrière-plan, le système vérifie l'unicité de l'email, hash le mot de passe via Werkzeug avant son stockage dans la base de données MySQL, et crée une nouvelle entrée dans la table users.

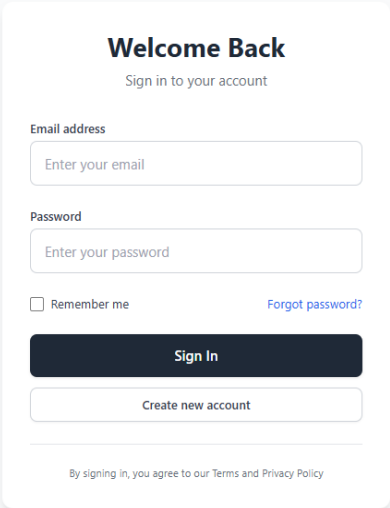
A 'Create Account' form with a dark blue header and a light gray background. It includes fields for 'Full Name', 'Email Address', and 'Password'. Below the password field is a note: 'Use 8 or more characters with a mix of letters, numbers & symbols'. There is a checkbox for 'I agree to the Terms and Privacy Policy'. A dark blue 'Create Account' button is prominent, followed by a link 'Already have an account? Sign in'. At the bottom, it says 'Secure authentication system'.

○ **Connexion (Login)**

La page de connexion, également stylisée avec Tailwind, offre:

- Une interface minimaliste et élégante
- Champs pour l'email et le mot de passe
- Option "Se souvenir de moi" utilisant des cookies
- Lien de récupération de mot de passe
- Messages d'erreur contextuels

Le processus de connexion utilise les sessions Flask pour maintenir l'état d'authentification de l'utilisateur à travers sa navigation.

A 'Welcome Back' login form with a dark blue header and a light gray background. It includes fields for 'Email address' and 'Password'. Below the password field is a checkbox for 'Remember me' and a link 'Forgot password?'. A dark blue 'Sign In' button is prominent, followed by a link 'Create new account'. At the bottom, it says 'By signing in, you agree to our Terms and Privacy Policy'.

- **Protection des Routes**

Toutes les routes sensibles sont protégées par un décorateur `@login_required` qui vérifie la présence et la validité de la session utilisateur. Ceci empêche l'accès non autorisé aux pages du tableau de bord, de création d'articles, et de profil.

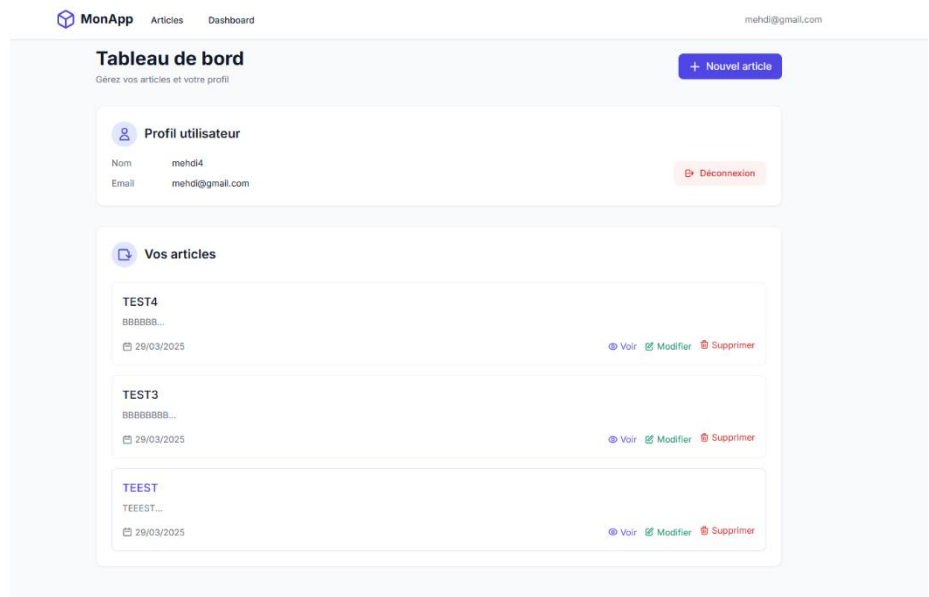
- **Gestion de Profil Utilisateur**

Le profil utilisateur offre une interface personnelle où l'utilisateur peut gérer ses informations:

- **Tableau de Bord Utilisateur**

Conçu avec Tailwind CSS, le tableau de bord présente:

- Une carte de profil avec le nom d'utilisateur et l'email
- Un design responsive s'adaptant à tous les écrans
- Des boutons d'action avec effets de survol et états actifs
- Des statistiques utilisateur (nombre d'articles publiés)
- Un bouton de déconnexion accessible



- **Gestion des Informations Personnelles**

L'interface de profil permet à l'utilisateur de:

- Visualiser ses informations actuelles
- Modifier son nom d'utilisateur
- Changer son adresse email (avec vérification)
- Mettre à jour son mot de passe de façon sécurisée

Les formulaires utilisent les classes utilitaires de Tailwind pour garantir cohérence et accessibilité, avec des transitions douces et des indicateurs d'état.

- **Système de Gestion d'Articles**

Le cœur fonctionnel de MonApp réside dans sa gestion d'articles, offrant une expérience complète de création et consommation de contenu:


3.1 Création d'Articles

L'interface de création d'articles propose:

- Un éditeur de texte intuitif
- Un champ de titre avec limite de caractères
- Zone de contenu extensible pour les articles longs
- Système d'upload d'images avec prévisualisation
- Boutons "Publier" et "Annuler" clairement distincts

L'intégration de Tailwind facilite la mise en place d'un design accessible et ergonomique, notamment:

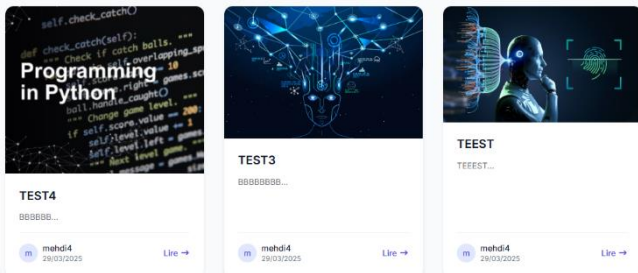
- Des indicateurs visuels de validation des champs
- Des zones de drop pour les images avec retour visuel
- Des états de focus accessibles au clavier
- Un design responsive s'adaptant aux mobiles et tablettes



3.2 Affichage des Articles

Articles

Découvrez nos derniers articles et actualités



La page d'articles présente une grille responsive de cartes, optimisée avec Tailwind pour:

- S'adapter automatiquement selon la taille d'écran (1, 2 ou 3 colonnes)
 - Afficher les images avec ratios constants
 - Tronquer intelligemment les textes trop longs
 - Présenter les métadonnées (auteur, date) de façon élégante
- Offrir des transitions douces au survol des articles

- **Visualisation d'Article Individuel**

Lorsqu'un utilisateur sélectionne un article, il accède à une page dédiée offrant:

- Une mise en page optimisée pour la lecture
- Un affichage de l'image en format optimal
- Des informations sur l'auteur et la date de publication

- **Modification et Suppression**

Pour les articles dont l'utilisateur est l'auteur, le système permet:

- L'édition complète du contenu et du titre
- Le remplacement ou la suppression de l'image
- La prévisualisation avant publication des modifications
- La suppression avec boîte de dialogue de confirmation

Chaque action est sécurisée par une vérification côté serveur de l'identité de l'utilisateur, empêchant toute manipulation non autorisée des articles.

- **Interface Responsive avec Tailwind CSS**

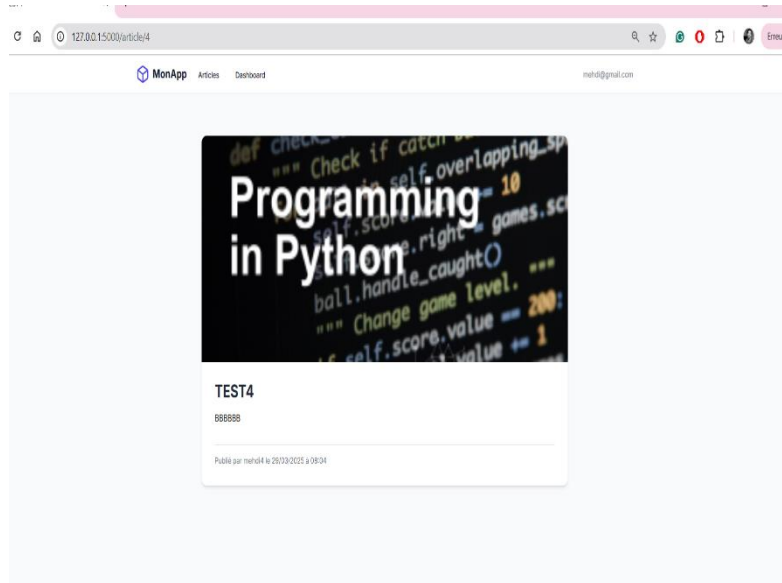
L'utilisation de Tailwind CSS comme framework d'interface a permis d'obtenir:

- **Design Cohérent**

- Système de couleurs unifié à travers l'application
- Typographie harmonisée avec hiérarchie visuelle claire
- Espacement et alignement constants grâce aux classes utilitaires
- Composants réutilisables (boutons, cartes, formulaires)

- **Configuration de base :**

[SQLALCHEMY_DATABASE_URI](#) : Définition de la base de données SQLite à utiliser.



SQLALCHEMY_TRACK_MODIFICATIONS : Désactive le suivi des modifications inutiles.

UPLOAD_FOLDER :

Dossier où les fichiers téléchargés seront stockés.

ALLOWED_EXTENSIONS

Liste des extensions de fichiers autorisées pour les images.

secret_key : Clé secrète utilisée pour les sessions.

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['UPLOAD_FOLDER'] = 'static/uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'}
app.secret_key = 'secret_key'
db = SQLAlchemy(app)
```

1.1 Modèles de Données

- User représente un utilisateur avec un identifiant unique, un nom, un email (unique) et un mot de passe.
- La relation articles établit une relation **un-à-plusieurs** entre un utilisateur et ses articles. Méthodes associées :
- **__init__** : Constructeur pour initialiser un utilisateur et hasher son mot de passe.
- **check_password** : Vérifie que le mot de passe fourni correspond au mot de passe haché dans la base de données.
- Article représente un article rédigé par un utilisateur avec un titre, un contenu, un nom de fichier d'image associé, une date de création et une relation avec l'utilisateur qui a écrit l'article

```
Codeium: Refactor | Explain
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    articles = db.relationship('Article', backref='author', lazy=True)

    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self, email, password, name):
        self.name = name
        self.email = email
        self.password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

    Codeium: Refactor | Explain | Generate Docstring | X
    def check_password(self, password):
        return bcrypt.checkpw(password.encode('utf-8'), self.password.encode('utf-8'))

# Modèle Article
Codeium: Refactor | Explain
class Article(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.Text, nullable=False)
    image_filename = db.Column(db.String(255))
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Fonctions utilitaires :

allowed_file(filename) : Vérifie si le fichier téléchargé a une extension valide (png, jpg, jpeg, gif).

```
Codeium: Refactor | Explain | Generate Docstring | X
def allowed_file(filename):
    return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']
```

is_logged_in() : Vérifie si un utilisateur est connecté (en vérifiant la session).

```
def is_logged_in():
    return 'email' in session
```

get_current_user() : Récupère l'utilisateur actuellement connecté à partir de la session.

```
Codeium: Refactor | Explain | Generate Docstring | X
def get_current_user():
    if is_logged_in():
        return User.query.filter_by(email=session['email']).first()
    return None
```

- **Routes :**

Route / : Redirige vers le tableau de bord si l'utilisateur est connecté, sinon vers la page de connexion.

```
@app.route('/')
def home():
    return redirect('/dashboard') if is_logged_in() else redirect('/login')
```

- **Route /login :**

Méthode HTTP : Cette route peut être appelée avec les méthodes HTTP GET et POST.

Vérification de la session :

- **is_logged_in()** est une fonction utilitaire qui vérifie si un utilisateur est actuellement connecté, c'est-à-dire si une clé 'email' est présente dans la session Flask. Si l'utilisateur est déjà connecté, il est redirigé vers le tableau de bord (/dashboard), sans avoir à se reconnecter.
- Lorsque le formulaire de connexion est soumis (méthode POST), les données du formulaire (email et mot de passe) sont récupérées à l'aide de **request.form**. Cela signifie que l'utilisateur a saisi son **email** et **mot de passe** dans le formulaire de connexion.

```
Codeium: Refactor | Explain | Generate Docstring | X
@app.route('/login', methods=['GET', 'POST'])
def login():
    if is_logged_in():
        return redirect('/dashboard')

    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password):
            session['email'] = user.email
            flash('Connexion réussie!', 'success')
            return redirect('/dashboard')
        else:
            flash('Email ou mot de passe incorrect', 'danger')

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

- **Vérification de l'utilisateur :**
- La requête **User.query.filter_by(email=email).first()** interroge la base de données pour trouver un utilisateur ayant l'email spécifié.
- Si un utilisateur est trouvé et que le mot de passe fourni par l'utilisateur correspond à celui stocké dans la base de données (grâce à la méthode **check_password** de la classe **User**), alors la connexion est réussie.
- **Méthode check_password :** Cette méthode compare le mot de passe saisi avec celui stocké dans la base de données (après avoir été haché avec **bcrypt**). Si les deux mots de passe correspondent, cela signifie que l'utilisateur a fourni le bon mot de passe.

Session utilisateur :

- Si les informations de connexion sont valides, l'email de l'utilisateur est stocké dans la session Flask (**session['email'] = user.email**), ce qui marque l'utilisateur comme connecté.
- Un message flash est affiché pour indiquer que la connexion a réussi avec **flash('Connexion réussie!', 'success')**. Le message "Connexion réussie!" apparaîtra avec une classe CSS de type **success** (typiquement en vert).
- Ensuite, l'utilisateur est redirigé vers le tableau de bord avec **return redirect('/dashboard')**.

- **Si les informations sont incorrectes :**
- Si l'email n'est pas trouvé ou si le mot de passe ne correspond pas, un message d'erreur est affiché à l'utilisateur avec **flash('Email ou mot de passe incorrect', 'danger')**. Ce message d'erreur apparaîtra avec une classe CSS de type danger (typiquement en rouge).

Route /register :

Méthode HTTP : Cette route accepte les requêtes HTTP de type GET (pour afficher le formulaire d'inscription) et POST (pour traiter les données envoyées par le formulaire d'inscription).

Lorsque l'utilisateur soumet le formulaire d'inscription (méthode POST), les données saisies dans le formulaire sont récupérées via **request.form**. Ces données comprennent :

name : Le nom de l'utilisateur.

email : L'email de l'utilisateur.

password : Le mot de passe que l'utilisateur veut utiliser.

Un nouvel objet **User** est créé avec les données du formulaire. Cependant, avant d'enregistrer cet utilisateur dans la base de données, le mot de passe doit être haché pour garantir la sécurité.

bcrypt est utilisé pour hacher le mot de passe.

bcrypt.hashpw() : Cette fonction prend le mot de passe sous forme de chaîne encodée (UTF-8), le "sale" avec un sel généré aléatoirement via **bcrypt.gensalt()**, puis retourne un mot de passe haché.

decode('utf-8') : Le résultat est converti en chaîne de caractères (au lieu d'un objet bytes) pour pouvoir être stocké dans la base de données.

L'objet new_user est ajouté à la session de la base de données via **db.session.add()**. Ensuite, **db.session.commit()** enregistre définitivement l'utilisateur dans la base de données.

Une fois l'utilisateur créé et ajouté à la base de données, il est redirigé vers la page de connexion **/login**, où il pourra se connecter avec les informations qu'il vient d'enregistrer.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        # handle request
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']

        new_user = User(name=name, email=email, password=password)
        db.session.add(new_user)
        db.session.commit()
        return redirect('/login')
    return render_template('register.html')
```

- **Tableau de bord utilisateur (Dashboard)**

Route : /dashboard

Fonction : dashboard()

Rôle : La fonction dashboard() utilise les fonctions suivantes :

- **is_logged_in()** : Vérifie si l'utilisateur est connecté en vérifiant la session.

- **get_current_user()** :
Récupère
l'utilisateur
actuellement
connecté à partir de
la session.

```
Codeium: Refactor | Explain | Generate Docs | X
@app.route('/dashboard')
def dashboard():
    if not is_logged_in():
        return redirect('/login')

    user = get_current_user()
    articles = Article.query.filter_by(user_id=user.id).order_by(Article.created_at.desc()).all()
    return render_template('dashboard.html', user=user, articles=articles)
```

- **Article.query.filter_by(user_id=user.id).order_by(Article.created_at.desc()).all()** : Récupère tous les articles de l'utilisateur, triés par date de création (du plus récent au plus ancien).
- **render_template('dashboard.html', user=user, articles=articles)** : Rend la page dashboard.html avec les informations de l'utilisateur et la liste de ses articles.

Si l'utilisateur n'est pas connecté, il est redirigé vers la page de connexion via `redirect('/login')`.

- **Gestion du CRUD pour les articles :**

Créer un article (Create)

- **Route** : /add_article

```
Codeium: Refactor | Explain | Generate Docs | X
@app.route('/add_article', methods=['GET', 'POST'])
def add_article():
    if not is_logged_in():
        return redirect('/login')

    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        user = get_current_user()

        # Gestion de l'upload de l'image
        image_filename = None
        if 'image' in request.files:
            file = request.files['image']
            if file and file.filename != '' and allowed_file(file.filename):
                filename = secure_filename(file.filename)
                file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                image_filename = filename

        new_article = Article(
            title=title,
            content=content,
            image_filename=image_filename,
            author=user
        )
        db.session.add(new_article)
        db.session.commit()

        flash('Article créé avec succès!', 'success')
        return redirect('/dashboard')

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

- **Fonction** : **add_article()**

- **Rôle** : Cette fonction permet à un utilisateur connecté de créer un nouvel article. Elle gère également le téléchargement d'une image associée à l'article (si présente).

- Lorsque l'utilisateur soumet un formulaire avec un titre, du contenu et une image (facultative), un nouvel objet **Article** est créé et ajouté à la base de données.

- L'article est ensuite sauvegardé en base avec **db.session.add(new_article)** et **db.session.commit()**.

- Enfin, l'utilisateur est redirigé vers le tableau de bord.

2. Lire les articles (Read)

- **Route** : /articles et /article/<int:article_id>

- **Fonction** :

- **articles()** : Affiche la liste de tous les articles, triés par date de création. Utilise **Article.query.order_by()** pour obtenir les articles.
- **article(article_id)** : Affiche un article spécifique basé sur son **article_id**. Utilise **Article.query.get_or_404()** pour récupérer un article en fonction de son ID. Si l'article n'est pas trouvé, une erreur 404 est renvoyée.

.Mettre à jour un article (Update)

- **Route** : /edit_article/<int:article_id>
- **Fonction** : `edit_article(article_id)`
- **Rôle** : Cette fonction permet à un utilisateur de modifier un article existant.
 - Si l'utilisateur est le propriétaire de l'article, il peut modifier son titre et son contenu.
 - Si une nouvelle image est téléchargée, l'ancienne image est supprimée et la nouvelle est sauvegardée.
 - L'article est ensuite mis à jour avec `db.session.commit()`.
 - Si l'utilisateur n'est pas le propriétaire de l'article, un message d'erreur est affiché et l'utilisateur est redirigé vers le tableau de bord.

Supprimer un article (Delete)

- **Route** : /delete_article/<int:article_id>
- **Fonction** : `delete_article(article_id)`
- **Rôle** : Cette fonction permet à un utilisateur de supprimer un article spécifique.
 - L'article est récupéré avec `Article.query.get_or_404()`.
 - Si l'utilisateur est le propriétaire de l'article, l'image associée à l'article est également supprimée (si présente) avant de supprimer l'article de la base de données.
 - La suppression est effectuée avec `db.session.delete(article)` et `db.session.commit()`.

Conclusion

Le projet **MonApp** développé avec **Python, Flask** et **SQLAlchemy** représente une solution complète et sécurisée pour la gestion d'articles en ligne, avec une expérience utilisateur fluide et responsive. Plusieurs points forts ressortent de ce projet :

- ✓ **Authentification sécurisée** : L'implémentation d'un système de connexion et d'inscription robuste, avec validation des informations et hachage des mots de passe, garantit la sécurité des données utilisateurs.
- ✓ **Interface responsive et ergonomique** : L'utilisation de **Tailwind CSS** permet d'obtenir un design moderne et adaptable, assurant une expérience agréable sur tous les appareils (ordinateurs, tablettes, smartphones).
- ✓ **Gestion de contenu optimisée** : Le système de création, modification et suppression d'articles permet aux utilisateurs de gérer facilement leurs contenus, tout en garantissant la sécurité des actions via des vérifications côté serveur.
- ✓ **Simplicité et fluidité d'utilisation** : L'interface de gestion de profil et le tableau de bord utilisateur permettent une navigation intuitive et un accès rapide aux fonctionnalités principales, rendant l'application facile à utiliser même pour les nouveaux utilisateurs.

- ✓ **Sécurisation des routes sensibles** : Grâce à l'utilisation du décorateur **@login_required**, toutes les routes critiques sont protégées, empêchant ainsi l'accès non autorisé aux pages sensibles de l'application.

En conclusion, **MonApp** est un projet complet qui allie performance, sécurité et design moderne. Il constitue une base solide pour développer des applications web nécessitant une gestion d'articles ou de contenus, tout en offrant une expérience utilisateur de qualité.