Projet Pratique Flask : Blog Complet

Présentation du Projet

Vous allez créer un **blog complet** avec toutes les fonctionnalités essentielles :

- ✅ Authentification (inscription, connexion, déconnexion)
- ✅ Gestion d'articles (CRUD complet)
- ✅ Commentaires sur les articles
- ✅ Catégories
- ✅ Recherche
- ✅ Interface moderne et responsive
- ✅ API REST

Structure du Projet

```
flask-blog/
│
├── app.py              # Application principale
├── models.py           # Modèles de base de données
├── forms.py            # Formulaires Flask-WTF
├── config.py           # Configuration
├── requirements.txt    # Dépendances
│
├── templates/
│   ├── base.html
│   ├── home.html
│   ├── post.html
│   ├── new_post.html
│   ├── edit_post.html
│   ├── login.html
```

```
│   ├── register.html
│   └── profile.html
│
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── img/
│
└── instance/
    └── blog.db          # Base de données SQLite
```

Étape 1 : Installation et Configuration

requirements.txt

```
Flask==3.0.0
Flask-SQLAlchemy==3.1.1
Flask-Login==0.6.3
Flask-WTF==1.2.1
WTForms==3.1.1
email-validator==2.1.0
python-dotenv==1.0.0
```

Installation

```
# Créer un environnement virtuel
python -m venv venv

# Activer (Windows)
venv\Scripts\activate
```

```
# Activer (Mac/Linux)
source venv/bin/activate

# Installer les dépendances
pip install -r requirements.txt
```

Étape 2 : Configuration

config.py

```
import os
from datetime import timedelta

class Config:
    """Configuration de base"""
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key-change-in-production'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///blog.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    # Flask-Login
    REMEMBER_COOKIE_DURATION = timedelta(days=7)

    # Pagination
    POSTS_PER_PAGE = 10
    COMMENTS_PER_PAGE = 20
```

Étape 3 : Modèles de Base de Données

models.py

```
from flask_sqlalchemy import SQLAlchemy
```

```
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime

db = SQLAlchemy()

class User(UserMixin, db.Model):
    """Modèle Utilisateur"""
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False, index=True)
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    password_hash = db.Column(db.String(200), nullable=False)
    bio = db.Column(db.Text)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    # Relations
    posts = db.relationship('Post', backref='author', lazy='dynamic', cascade='all, delete-orphan')
    comments = db.relationship('Comment', backref='author', lazy='dynamic', cascade='all, delete-orphan')

    def set_password(self, password):
        """Hash le mot de passe"""
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        """Vérifie le mot de passe"""
```

```
        return check_password_hash(self.password_hash, password)

    def __repr__(self):
        return f'<User {self.username}>'


class Category(db.Model):
    """Modèle Catégorie"""
    __tablename__ = 'categories'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True, nullable=False)
    slug = db.Column(db.String(50), unique=True, nullable=False)

    # Relations
    posts = db.relationship('Post', backref='category', lazy='dynamic')

    def __repr__(self):
        return f'<Category {self.name}>'


class Post(db.Model):
    """Modèle Article"""
    __tablename__ = 'posts'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    slug = db.Column(db.String(200), unique=True, nullable=False)
```

```
    content = db.Column(db.Text, nullable=False)
    excerpt = db.Column(db.String(500))

    # Timestamps
    created_at = db.Column(db.DateTime, default=datetime.utcnow, index=True)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)

    # Clés étrangères
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.id'))

    # Relations
    comments = db.relationship('Comment', backref='post', lazy='dynamic',
                               cascade='all, delete-orphan')

    def __repr__(self):
        return f'<Post {self.title}>'


class Comment(db.Model):
    """Modèle Commentaire"""
    __tablename__ = 'comments'

    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
    # Clés étrangères
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    post_id = db.Column(db.Integer, db.ForeignKey('posts.id'), nullable=False)

    def __repr__(self):
        return f'<Comment by {self.author.username} on {self.post.title}>'
```

Étape 4 : Formulaires

forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, TextAreaField, SelectField, SubmitField
from wtforms.validators import DataRequired, Email, Length, EqualTo, ValidationError
from models import User

class RegistrationForm(FlaskForm):
    """Formulaire d'inscription"""
    username = StringField('Nom d\'utilisateur',
                           validators=[DataRequired(), Length(min=3, max=80)])
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    password = PasswordField('Mot de passe',
                             validators=[DataRequired(), Length(min=8)])
    confirm_password = PasswordField('Confirmer le mot de passe',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('S\'inscrire')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
```

```
        if user:
            raise ValidationError('Ce nom d\'utilisateur est déjà pris.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('Cet email est déjà utilisé.')


class LoginForm(FlaskForm):
    """Formulaire de connexion"""
    username = StringField('Nom d\'utilisateur',
                           validators=[DataRequired()])
    password = PasswordField('Mot de passe',
                             validators=[DataRequired()])
    submit = SubmitField('Se connecter')


class PostForm(FlaskForm):
    """Formulaire d'article"""
    title = StringField('Titre',
                        validators=[DataRequired(), Length(min=5, max=200)])
    content = TextAreaField('Contenu',
                            validators=[DataRequired(), Length(min=20)])
    excerpt = StringField('Extrait',
                          validators=[Length(max=500)])
    category = SelectField('Catégorie', coerce=int)
    submit = SubmitField('Publier')
```

```
class CommentForm(FlaskForm):
    """Formulaire de commentaire"""
    content = TextAreaField('Commentaire',
                            validators=[DataRequired(), Length(min=2, max=500)])
    submit = SubmitField('Commenter')


class ProfileForm(FlaskForm):
    """Formulaire de profil"""
    username = StringField('Nom d\'utilisateur',
                           validators=[DataRequired(), Length(min=3, max=80)])
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    bio = TextAreaField('Biographie',
                        validators=[Length(max=500)])
    submit = SubmitField('Mettre à jour')
```

Étape 5 : Application Principale

app.py

```
from flask import Flask, render_template, redirect, url_for, flash, request, abort
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from werkzeug.utils import secure_filename
from slugify import slugify
import os
```

```
from config import Config
from models import db, User, Post, Category, Comment
from forms import RegistrationForm, LoginForm, PostForm, CommentForm, ProfileForm

# Initialisation de l'application
app = Flask(__name__)
app.config.from_object(Config)

# Initialisation des extensions
db.init_app(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message = 'Veuillez vous connecter pour accéder à cette page.'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


# ==================== ROUTES PRINCIPALES ====================

@app.route('/')
def home():
    """Page d'accueil avec liste des articles"""
    page = request.args.get('page', 1, type=int)
    posts = Post.query.order_by(Post.created_at.desc()).paginate(
        page=page, per_page=app.config['POSTS_PER_PAGE'], error_out=False
    )
```

```
    return render_template('home.html', posts=posts)


@app.route('/post/<int:post_id>')
def post(post_id):
    """Page détail d'un article"""
    post = Post.query.get_or_404(post_id)
    form = CommentForm()

    page = request.args.get('page', 1, type=int)
    comments = post.comments.order_by(Comment.created_at.desc()).paginate(
        page=page, per_page=app.config['COMMENTS_PER_PAGE'], error_out=False
    )

    return render_template('post.html', post=post, form=form, comments=comments)


@app.route('/post/<int:post_id>/comment', methods=['POST'])
@login_required
def add_comment(post_id):
    """Ajouter un commentaire"""
    post = Post.query.get_or_404(post_id)
    form = CommentForm()

    if form.validate_on_submit():
        comment = Comment(
            content=form.content.data,
            user_id=current_user.id,
```

```
            post_id=post.id
        )
        db.session.add(comment)
        db.session.commit()

        flash('Commentaire ajouté avec succès !', 'success')

    return redirect(url_for('post', post_id=post.id))


# ==================== GESTION DES ARTICLES ====================

@app.route('/post/new', methods=['GET', 'POST'])
@login_required
def new_post():
    """Créer un nouvel article"""
    form = PostForm()
    form.category.choices = [(c.id, c.name) for c in Category.query.all()]
    form.category.choices.insert(0, (0, '-- Sélectionner une catégorie --'))

    if form.validate_on_submit():
        # Générer un slug unique
        slug = slugify(form.title.data)
        counter = 1
        while Post.query.filter_by(slug=slug).first():
            slug = f"{slugify(form.title.data)}-{counter}"
            counter += 1
```

```
        post = Post(
            title=form.title.data,
            slug=slug,
            content=form.content.data,
            excerpt=form.excerpt.data,
            user_id=current_user.id,
            category_id=form.category.data if form.category.data != 0 else None
        )

        db.session.add(post)
        db.session.commit()

        flash('Article publié avec succès !', 'success')
        return redirect(url_for('post', post_id=post.id))

    return render_template('new_post.html', form=form, title='Nouvel Article')


@app.route('/post/<int:post_id>/edit', methods=['GET', 'POST'])
@login_required
def edit_post(post_id):
    """Modifier un article"""
    post = Post.query.get_or_404(post_id)

    # Vérifier que l'utilisateur est l'auteur
    if post.author != current_user:
        abort(403)
```

```
    form = PostForm()
    form.category.choices = [(c.id, c.name) for c in Category.query.all()]
    form.category.choices.insert(0, (0, '-- Sélectionner une catégorie --'))

    if form.validate_on_submit():
        post.title = form.title.data
        post.content = form.content.data
        post.excerpt = form.excerpt.data
        post.category_id = form.category.data if form.category.data != 0 else None

        db.session.commit()

        flash('Article mis à jour !', 'success')
        return redirect(url_for('post', post_id=post.id))

    elif request.method == 'GET':
        form.title.data = post.title
        form.content.data = post.content
        form.excerpt.data = post.excerpt
        form.category.data = post.category_id if post.category_id else 0

    return render_template('edit_post.html', form=form, post=post, title='Modifier
l\'Article')



@app.route('/post/<int:post_id>/delete', methods=['POST'])
@login_required
def delete_post(post_id):
```

```
    """Supprimer un article"""
    post = Post.query.get_or_404(post_id)

    if post.author != current_user:
        abort(403)

    db.session.delete(post)
    db.session.commit()

    flash('Article supprimé.', 'info')
    return redirect(url_for('home'))


# ==================== AUTHENTIFICATION ====================

@app.route('/register', methods=['GET', 'POST'])
def register():
    """Inscription"""
    if current_user.is_authenticated:
        return redirect(url_for('home'))

    form = RegistrationForm()

    if form.validate_on_submit():
        user = User(
            username=form.username.data,
            email=form.email.data
        )
```

```
        user.set_password(form.password.data)

        db.session.add(user)
        db.session.commit()

        flash('Compte créé avec succès ! Vous pouvez maintenant vous connecter.',
'success')
        return redirect(url_for('login'))

    return render_template('register.html', form=form, title='Inscription')


@app.route('/login', methods=['GET', 'POST'])
def login():
    """Connexion"""
    if current_user.is_authenticated:
        return redirect(url_for('home'))

    form = LoginForm()

    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()

        if user and user.check_password(form.password.data):
            login_user(user, remember=True)

            next_page = request.args.get('next')
            flash('Connexion réussie !', 'success')
```

```
            return redirect(next_page or url_for('home'))

        flash('Nom d\'utilisateur ou mot de passe incorrect.', 'error')

    return render_template('login.html', form=form, title='Connexion')


@app.route('/logout')
@login_required
def logout():
    """Déconnexion"""
    logout_user()
    flash('Vous êtes déconnecté.', 'info')
    return redirect(url_for('home'))


# ==================== PROFIL ====================

@app.route('/profile/<username>')
def profile(username):
    """Page profil utilisateur"""
    user = User.query.filter_by(username=username).first_or_404()
    page = request.args.get('page', 1, type=int)
    posts = user.posts.order_by(Post.created_at.desc()).paginate(
        page=page, per_page=5, error_out=False
    )
    return render_template('profile.html', user=user, posts=posts)
```

```
@app.route('/profile/edit', methods=['GET', 'POST'])
@login_required
def edit_profile():
    """Modifier le profil"""
    form = ProfileForm()

    if form.validate_on_submit():
        current_user.username = form.username.data
        current_user.email = form.email.data
        current_user.bio = form.bio.data

        db.session.commit()

        flash('Profil mis à jour !', 'success')
        return redirect(url_for('profile', username=current_user.username))

    elif request.method == 'GET':
        form.username.data = current_user.username
        form.email.data = current_user.email
        form.bio.data = current_user.bio

    return render_template('edit_profile.html', form=form, title='Modifier le Profil')


# ==================== CATÉGORIES ====================

@app.route('/category/<slug>')
```

```
def category(slug):
    """Articles par catégorie"""
    category = Category.query.filter_by(slug=slug).first_or_404()
    page = request.args.get('page', 1, type=int)
    posts = category.posts.order_by(Post.created_at.desc()).paginate(
        page=page, per_page=app.config['POSTS_PER_PAGE'], error_out=False
    )
    return render_template('category.html', category=category, posts=posts)


# ==================== RECHERCHE ====================

@app.route('/search')
def search():
    """Recherche d'articles"""
    query = request.args.get('q', '')
    page = request.args.get('page', 1, type=int)

    if query:
        posts = Post.query.filter(
            Post.title.contains(query) | Post.content.contains(query)
        ).order_by(Post.created_at.desc()).paginate(
            page=page, per_page=app.config['POSTS_PER_PAGE'], error_out=False
        )
    else:
        posts = None

    return render_template('search.html', posts=posts, query=query)
```

```
# ==================== GESTION D'ERREURS ====================

@app.errorhandler(404)
def not_found_error(error):
    return render_template('404.html'), 404

@app.errorhandler(403)
def forbidden_error(error):
    return render_template('403.html'), 403

@app.errorhandler(500)
def internal_error(error):
    db.session.rollback()
    return render_template('500.html'), 500


# ==================== CONTEXT PROCESSORS ====================

@app.context_processor
def inject_categories():
    """Rendre les catégories disponibles dans tous les templates"""
    categories = Category.query.all()
    return dict(categories=categories)


# ==================== INITIALISATION ====================
```

```
def init_db():
    """Initialiser la base de données avec des données de test"""
    with app.app_context():
        db.create_all()

        # Créer des catégories si elles n'existent pas
        if Category.query.count() == 0:
            categories = [
                Category(name='Python', slug='python'),
                Category(name='Web Development', slug='web-development'),
                Category(name='Data Science', slug='data-science'),
                Category(name='Machine Learning', slug='machine-learning'),
            ]
            for cat in categories:
                db.session.add(cat)

            db.session.commit()
            print('✓ Catégories créées')

        # Créer un utilisateur de test
        if User.query.count() == 0:
            user = User(username='admin', email='admin@example.com')
            user.set_password('password123')
            db.session.add(user)
            db.session.commit()
            print('✓ Utilisateur admin créé (password123)')
```

```
if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

(Suite dans un document séparé avec les templates...)

Installation de slugify

pip install python-slugify

Ajoutez dans requirements.txt :

python-slugify==8.0.1

Lancer l'Application

```
# 1. Créer l'environnement virtuel et installer les dépendances
python -m venv venv
source venv/bin/activate  # ou venv\Scripts\activate sur Windows
pip install -r requirements.txt

# 2. Lancer l'application
python app.py

# 3. Accéder au site
# http://127.0.0.1:5000

# 4. Se connecter avec :
# Username: admin
# Password: password123
```

Fonctionnalités à Implémenter (Exercices)

1. **Pagination améliorée** avec numéros de pages
2. **Upload d'images** pour les articles
3. **Tags** pour les articles (many-to-many)
4. **Like/Unlike** sur les articles
5. **Notifications** pour les nouveaux commentaires
6. **Flux RSS**
7. **API REST** complète
8. **Admin panel** pour gérer les utilisateurs
9. **Markdown** pour le contenu des articles
10. **Recherche avancée** avec filtres

Ce projet vous donne une base solide pour créer n'importe quelle application web avec Flask ! 🚀