Cours Flask - Partie 2 : Concepts Avancés

7. Sessions et Authentification

Qu'est-ce qu'une Session ?

Une session permet de stocker des informations sur l'utilisateur entre différentes requêtes HTTP. Par défaut, HTTP est "stateless" (sans état).

Sessions de Base avec Flask

```
from flask import Flask, session, redirect, url_for, request


app = Flask(__name__)

app.secret_key = 'votre_cle_secrete_tres_securisee'


@app.route('/')

def home():

  if 'username' in session:

    return f'Bienvenue {session["username"]} !'

  return 'Vous n\'êtes pas connecté'


@app.route('/login', methods=['GET', 'POST'])

def login():

  if request.method == 'POST':

    # Stocker l'username dans la session

    session['username'] = request.form['username']

    session['user_id'] = 1  # ID de l'utilisateur

    return redirect(url_for('home'))


  return '''

    <form method="post">

      <input type="text" name="username" placeholder="Username">
```

```
        <button type="submit">Se connecter</button>
    </form>
    '''

@app.route('/logout')
def logout():
    # Supprimer l'username de la session
    session.pop('username', None)
    session.pop('user_id', None)
    return redirect(url_for('home'))
```

Système d'Authentification Complet

Installation des dépendances

```
pip install flask-login werkzeug
```

Modèle User avec Hash de Mot de Passe

```
from flask import Flask, render_template, redirect, url_for, flash, request
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, logout_user,
login_required, current_user
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SECRET_KEY'] = 'your-secret-key'

db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
```

```
# Modèle User
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(200), nullable=False)

    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)

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

Routes d'Authentification

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

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

```
        # Vérifier si l'utilisateur existe déjà
        if User.query.filter_by(username=username).first():
            flash('Ce nom d\'utilisateur existe déjà', 'error')
            return redirect(url_for('register'))

        if User.query.filter_by(email=email).first():
            flash('Cet email est déjà utilisé', 'error')
            return redirect(url_for('register'))

        # Créer le nouvel utilisateur
        user = User(username=username, email=email)
        user.set_password(password)

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

        flash('Inscription réussie ! Vous pouvez vous connecter.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')

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

    if request.method == 'POST':
```

```
        username = request.form['username']
        password = request.form['password']
        remember = request.form.get('remember', False)

        user = User.query.filter_by(username=username).first()

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

            # Rediriger vers la page demandée ou l'accueil
            next_page = request.args.get('next')
            return redirect(next_page or url_for('home'))

        flash('Identifiants incorrects', 'error')

    return render_template('login.html')

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

@app.route('/profile')
@login_required
def profile():
    return render_template('profile.html', user=current_user)
```

Templates d'Authentification

templates/register.html

```
{% extends "base.html" %}

{% block content %}
<h2>Inscription</h2>

<form method="POST">
  <div>
    <label>Nom d'utilisateur:</label>
    <input type="text" name="username" required>
  </div>

  <div>
    <label>Email:</label>
    <input type="email" name="email" required>
  </div>

  <div>
    <label>Mot de passe:</label>
    <input type="password" name="password" required minlength="8">
  </div>

  <button type="submit">S'inscrire</button>
</form>

<p>Déjà un compte ? <a href="{{ url_for('login') }}">Connectez-vous</a></p>
{% endblock %}
```

templates/login.html

```
{% extends "base.html" %}

{% block content %}
<h2>Connexion</h2>

<form method="POST">
  <div>
    <label>Nom d'utilisateur:</label>
    <input type="text" name="username" required>
  </div>

  <div>
    <label>Mot de passe:</label>
    <input type="password" name="password" required>
  </div>

  <div>
    <input type="checkbox" name="remember" value="1">
    <label>Se souvenir de moi</label>
  </div>

  <button type="submit">Se connecter</button>
</form>

<p>Pas encore de compte ? <a href="{{ url_for('register') }}">Inscrivez-vous</a></p>
{% endblock %}
```

templates/base.html (avec navigation authentifiée)

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Mon Site{% endblock %}</title>
</head>
<body>
  <nav>
    <a href="{{ url_for('home') }}">Accueil</a>

    {% if current_user.is_authenticated %}
      <a href="{{ url_for('profile') }}">Profil</a>
      <a href="{{ url_for('logout') }}">Déconnexion</a>
      <span>Bienvenue {{ current_user.username }}</span>
    {% else %}
      <a href="{{ url_for('login') }}">Connexion</a>
      <a href="{{ url_for('register') }}">Inscription</a>
    {% endif %}
  </nav>

  {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
      {% for category, message in messages %}
        <div class="flash-{{ category }}">{{ message }}</div>
      {% endfor %}
    {% endif %}
  {% endwith %}
```

```
  <main>
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

Protection de Routes

```
from flask_login import login_required, current_user
from functools import wraps

# Route protégée simple
@app.route('/dashboard')
@login_required
def dashboard():
  return render_template('dashboard.html')

# Vérification personnalisée
def admin_required(f):
  @wraps(f)
  def decorated_function(*args, **kwargs):
    if not current_user.is_authenticated:
      return redirect(url_for('login'))
    if not current_user.is_admin:  # Supposons qu'il y a un champ is_admin
      flash('Accès refusé', 'error')
      return redirect(url_for('home'))
    return f(*args, **kwargs)
  return decorated_function

@app.route('/admin')
```

```
@admin_required
def admin_panel():
    return render_template('admin.html')
```

Exercice 7 : Système d'Authentification Complet

Consigne : Créez un réseau social simplifié :

1. **Système d'authentification** :
    - Inscription avec validation (username unique, email valide, mot de passe 8+ caractères)
    - Connexion avec "Se souvenir de moi"
    - Déconnexion
2. **Profils utilisateurs** :
    - Page profil avec bio, photo, date d'inscription
    - Édition du profil (protégé)
    - Changement de mot de passe
3. **Sécurité** :
    - Hasher les mots de passe
    - Routes protégées avec @login_required
    - Vérifier que l'utilisateur ne peut modifier que son propre profil
4. **Fonctionnalités** :
    - Liste des utilisateurs
    - Voir le profil de n'importe quel utilisateur
    - Compteur de posts par utilisateur

---

8. Fichiers Statiques (CSS, JavaScript, Images)

Structure des Fichiers Statiques

```
mon_projet/
├── app.py
├── templates/
```

```
│   └── home.html
└── static/
    ├── css/
    │   └── style.css
    ├── js/
    │   └── script.js
    └── img/
        └── logo.png
```

Lier des Fichiers Statiques

templates/base.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Mon Site{% endblock %}</title>

    <!-- CSS -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

    <!-- Favicon -->
    <link rel="icon" href="{{ url_for('static', filename='img/favicon.ico') }}">
</head>
<body>
    <header>
        <img src="{{ url_for('static', filename='img/logo.png') }}" alt="Logo">
        <nav>
            <a href="{{ url_for('home') }}">Accueil</a>
```

```
        <a href="{{ url_for('about') }}">À propos</a>
      </nav>
    </header>

    <main>
      {% block content %}{% endblock %}
    </main>

    <footer>
      <p>&copy; 2024 Mon Site</p>
    </footer>

    <!-- JavaScript -->
    <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Exemple de CSS

static/css/style.css

```
/* Reset et Base */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Arial', sans-serif;
    line-height: 1.6;
```

```
    color: #333;
    background-color: #f4f4f4;
}

/* Header */
header {
    background: #333;
    color: #fff;
    padding: 1rem 0;
    text-align: center;
}

header img {
    height: 50px;
    margin-bottom: 10px;
}

nav a {
    color: #fff;
    text-decoration: none;
    padding: 0 15px;
    transition: color 0.3s;
}

nav a:hover {
    color: #f4f4f4;
}
```

```
/* Main Content */
main {
  max-width: 1200px;
  margin: 2rem auto;
  padding: 0 20px;
  background: #fff;
  padding: 2rem;
  border-radius: 5px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

/* Forms */
form {
  max-width: 500px;
  margin: 0 auto;
}

form div {
  margin-bottom: 1rem;
}

label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: bold;
}

input[type="text"],
```

```
input[type="email"],
input[type="password"],
textarea {
  width: 100%;
  padding: 0.5rem;
  border: 1px solid #ddd;
  border-radius: 4px;
}

button {
  background: #333;
  color: #fff;
  padding: 0.7rem 2rem;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background 0.3s;
}

button:hover {
  background: #555;
}

/* Flash Messages */
.flash-success {
  background: #d4edda;
  color: #155724;
  padding: 1rem;
```

```
    border-radius: 4px;
    margin-bottom: 1rem;
    border: 1px solid #c3e6cb;
}

.flash-error {
    background: #f8d7da;
    color: #721c24;
    padding: 1rem;
    border-radius: 4px;
    margin-bottom: 1rem;
    border: 1px solid #f5c6cb;
}

/* Cards */
.card {
    background: #fff;
    padding: 1.5rem;
    margin-bottom: 1rem;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.card h3 {
    margin-bottom: 0.5rem;
}

/* Footer */
```

```
footer {
  background: #333;
  color: #fff;
  text-align: center;
  padding: 1rem 0;
  margin-top: 2rem;
}
```

Exemple de JavaScript

static/js/script.js

```
// Confirmation avant suppression
document.querySelectorAll('.delete-btn').forEach(button => {
  button.addEventListener('click', function(e) {
    if (!confirm('Êtes-vous sûr de vouloir supprimer cet élément ?')) {
      e.preventDefault();
    }
  });
});

// Validation de formulaire
document.querySelector('#contact-form')?.addEventListener('submit', function(e) {
  const email = document.querySelector('#email').value;
  const message = document.querySelector('#message').value;

  if (!email.includes('@')) {
    alert('Email invalide');
    e.preventDefault();
    return;
  }
```

```
    if (message.length < 10) {
        alert('Le message doit contenir au moins 10 caractères');
        e.preventDefault();
        return;
    }
});

// Animation au scroll
window.addEventListener('scroll', function() {
    const header = document.querySelector('header');
    if (window.scrollY > 100) {
        header.style.boxShadow = '0 2px 10px rgba(0,0,0,0.1)';
    } else {
        header.style.boxShadow = 'none';
    }
});

// Messages flash auto-dismiss
setTimeout(() => {
    const flashes = document.querySelectorAll('[class^="flash-"]');
    flashes.forEach(flash => {
        flash.style.transition = 'opacity 0.5s';
        flash.style.opacity = '0';
        setTimeout(() => flash.remove(), 500);
    });
}, 5000);
```

Intégration de Frameworks CSS

Bootstrap

```
<head>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
      <a class="navbar-brand" href="/">Mon Site</a>
    </div>
  </nav>

  <div class="container mt-5">
    {% block content %}{% endblock %}
  </div>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
```

Tailwind CSS

```
<head>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100">
  <nav class="bg-gray-800 text-white p-4">
    <div class="container mx-auto">
```

```
        <a href="/" class="text-xl font-bold">Mon Site</a>
      </div>
    </nav>

    <main class="container mx-auto mt-8 px-4">
      {% block content %}{% endblock %}
    </main>
</body>
```

Servir des Fichiers Uploadés

```
from flask import send_from_directory
import os

UPLOAD_FOLDER = 'uploads'

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

# Dans un template
# <img src="{{ url_for('uploaded_file', filename=user.avatar) }}">
```

Exercice 8 : Interface Moderne

Consigne : Créez un blog avec une interface moderne :

1. **Design responsive** avec Bootstrap ou Tailwind
2. **Navigation** :
    - Menu avec logo
    - Liens vers Accueil, Articles, À propos, Contact
    - Indicateur de connexion (username si connecté)
3. **Page d'accueil** :

- Hero section avec image de fond
- Grille de cards pour les articles (3 colonnes)
- Sidebar avec catégories et articles populaires

4. **JavaScript** :
   - Confirmation avant suppression
   - Validation de formulaire côté client
   - Animation au scroll
   - Messages flash qui disparaissent après 5 secondes
5. **Images** :
   - Logo du site
   - Image de couverture pour chaque article
   - Avatar pour les utilisateurs
6. Optimisez les performances (minification, compression)

---

9. API REST avec Flask

Qu'est-ce qu'une API REST ?

REST (Representational State Transfer) est une architecture pour créer des APIs.

Principes REST :

- Utilisation des méthodes HTTP (GET, POST, PUT, DELETE)
- URLs logiques et hiérarchiques
- Format de données JSON
- Sans état (stateless)

API Simple

```
from flask import Flask, jsonify, request


app = Flask(__name__)


# Données exemple
```

```
books = [
  {'id': 1, 'title': '1984', 'author': 'George Orwell'},
  {'id': 2, 'title': 'Le Seigneur des Anneaux', 'author': 'J.R.R. Tolkien'}
]

# GET : Récupérer tous les livres
@app.route('/api/books', methods=['GET'])
def get_books():
  return jsonify(books)

# GET : Récupérer un livre spécifique
@app.route('/api/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
  book = next((b for b in books if b['id'] == book_id), None)
  if book:
    return jsonify(book)
  return jsonify({'error': 'Livre non trouvé'}), 404

# POST : Créer un nouveau livre
@app.route('/api/books', methods=['POST'])
def create_book():
  data = request.get_json()

  new_book = {
    'id': len(books) + 1,
    'title': data['title'],
    'author': data['author']
  }
```

```
    books.append(new_book)

    return jsonify(new_book), 201

# PUT : Mettre à jour un livre
@app.route('/api/books/<int:book_id>', methods=['PUT'])
def update_book(book_id):
    book = next((b for b in books if b['id'] == book_id), None)
    if not book:
        return jsonify({'error': 'Livre non trouvé'}), 404

    data = request.get_json()
    book['title'] = data.get('title', book['title'])
    book['author'] = data.get('author', book['author'])

    return jsonify(book)

# DELETE : Supprimer un livre
@app.route('/api/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    global books
    books = [b for b in books if b['id'] != book_id]
    return jsonify({'message': 'Livre supprimé'}), 200
```

Tester l'API

Avec curl (terminal)

```
# GET
curl http://localhost:5000/api/books
```

```
# POST
curl -X POST http://localhost:5000/api/books \
  -H "Content-Type: application/json" \
  -d '{"title":"Nouveau Livre","author":"Auteur"}'

# PUT
curl -X PUT http://localhost:5000/api/books/1 \
  -H "Content-Type: application/json" \
  -d '{"title":"Titre Modifié"}'

# DELETE
curl -X DELETE http://localhost:5000/api/books/1
```

Avec Python (requests)

```
import requests

BASE_URL = 'http://localhost:5000/api'

# GET
response = requests.get(f'{BASE_URL}/books')
print(response.json())

# POST
data = {'title': 'Nouveau Livre', 'author': 'Auteur'}
response = requests.post(f'{BASE_URL}/books', json=data)
print(response.json())

# PUT
data = {'title': 'Titre Modifié'}
```

```
response = requests.put(f'{BASE_URL}/books/1', json=data)
print(response.json())

# DELETE
response = requests.delete(f'{BASE_URL}/books/1')
print(response.json())
```

API avec Base de Données

```
from flask import Flask, jsonify, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///api.db'
db = SQLAlchemy(app)

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    author = db.Column(db.String(100), nullable=False)
    year = db.Column(db.Integer)

    def to_dict(self):
        return {
            'id': self.id,
            'title': self.title,
            'author': self.author,
            'year': self.year
        }
```

```
@app.route('/api/books', methods=['GET'])
def get_books():
    books = Book.query.all()
    return jsonify([book.to_dict() for book in books])

@app.route('/api/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    book = Book.query.get_or_404(book_id)
    return jsonify(book.to_dict())

@app.route('/api/books', methods=['POST'])
def create_book():
    data = request.get_json()

    book = Book(
        title=data['title'],
        author=data['author'],
        year=data.get('year')
    )

    db.session.add(book)
    db.session.commit()

    return jsonify(book.to_dict()), 201

@app.route('/api/books/<int:book_id>', methods=['PUT'])
def update_book(book_id):
    book = Book.query.get_or_404(book_id)
```

```
    data = request.get_json()

    book.title = data.get('title', book.title)
    book.author = data.get('author', book.author)
    book.year = data.get('year', book.year)

    db.session.commit()

    return jsonify(book.to_dict())

@app.route('/api/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    book = Book.query.get_or_404(book_id)
    db.session.delete(book)
    db.session.commit()

    return jsonify({'message': 'Livre supprimé'}), 200
```

Gestion d'Erreurs API

```
@app.errorhandler(404)
def not_found(error):
    return jsonify({'error': 'Ressource non trouvée'}), 404

@app.errorhandler(400)
def bad_request(error):
    return jsonify({'error': 'Requête invalide'}), 400

@app.errorhandler(500)
def internal_error(error):
```

```
    return jsonify({'error': 'Erreur serveur'}), 500

# Validation personnalisée
@app.route('/api/books', methods=['POST'])
def create_book():
    data = request.get_json()

    if not data:
        return jsonify({'error': 'Aucune donnée fournie'}), 400

    if 'title' not in data or 'author' not in data:
        return jsonify({'error': 'Titre et auteur requis'}), 400

    # Créer le livre...
```

CORS (Cross-Origin Resource Sharing)

Pour permettre aux applications frontend d'utiliser votre API :

```
pip install flask-cors
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # Permet toutes les origines

# Ou configuration spécifique
CORS(app, resources={
    r"/api/*": {
        "origins": ["http://localhost:3000", "https://monsite.com"],
        "methods": ["GET", "POST", "PUT", "DELETE"],
        "allow_headers": ["Content-Type"]
```

```
    }
})
```

Pagination

```
@app.route('/api/books', methods=['GET'])
def get_books():
    page = request.args.get('page', 1, type=int)
    per_page = request.args.get('per_page', 10, type=int)

    books_paginated = Book.query.paginate(
        page=page,
        per_page=per_page,
        error_out=False
    )

    return jsonify({
        'books': [book.to_dict() for book in books_paginated.items],
        'total': books_paginated.total,
        'pages': books_paginated.pages,
        'current_page': page,
        'has_next': books_paginated.has_next,
        'has_prev': books_paginated.has_prev
    })
```

Authentification API (JWT)

```
pip install pyjwt
import jwt
from datetime import datetime, timedelta
from functools import wraps
```

```
SECRET_KEY = 'votre_cle_secrete'

def generate_token(user_id):
    """Génère un token JWT"""
    payload = {
        'user_id': user_id,
        'exp': datetime.utcnow() + timedelta(hours=24)
    }
    return jwt.encode(payload, SECRET_KEY, algorithm='HS256')

def token_required(f):
    """Décorateur pour protéger les routes API"""
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')

        if not token:
            return jsonify({'error': 'Token manquant'}), 401

        try:
            # Enlever "Bearer " du token
            token = token.split()[1] if ' ' in token else token
            data = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
            current_user_id = data['user_id']
        except:
            return jsonify({'error': 'Token invalide'}), 401

        return f(current_user_id, *args, **kwargs)
```

```
    return decorated

@app.route('/api/login', methods=['POST'])
def api_login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    # Vérifier les identifiants (simplifié)
    user = User.query.filter_by(username=username).first()
    if user and user.check_password(password):
        token = generate_token(user.id)
        return jsonify({'token': token})

    return jsonify({'error': 'Identifiants incorrects'}), 401

@app.route('/api/protected', methods=['GET'])
@token_required
def protected_route(current_user_id):
    return jsonify({
        'message': 'Accès autorisé',
        'user_id': current_user_id
    })
```

Exercice 9 : API REST Complète

Consigne : Créez une API pour une application de gestion de tâches :

1. **Endpoints** :
    - GET /api/tasks : Liste toutes les tâches (avec pagination)

- GET /api/tasks/<id> : Détails d'une tâche
- POST /api/tasks : Créer une tâche
- PUT /api/tasks/<id> : Modifier une tâche
- DELETE /api/tasks/<id> : Supprimer une tâche
- GET /api/tasks?status=completed : Filtrer par statut

2. **Modèle Task** :
   - id, title, description, status (pending/completed), priority (low/medium/high), created_at, due_date
3. **Authentification** :
   - POST /api/register : Inscription
   - POST /api/login : Connexion (retourne un JWT)
   - Routes protégées par JWT
4. **Validation** :
   - Vérifier les données entrantes
   - Messages d'erreur clairs
5. **Documentation** :
   - Documentez chaque endpoint (paramètres, réponses)
6. **Tests** :
   - Créez un script Python qui teste tous les endpoints

---

10. Déploiement d'une Application Flask

Préparation au Déploiement

requirements.txt

pip freeze > requirements.txt

Contenu typique :

Flask==3.0.0

Flask-SQLAlchemy==3.1.1

Flask-Login==0.6.3

Flask-WTF==1.2.1

python-dotenv==1.0.0

Variables d'Environnement

Fichier .env

FLASK_APP=app.py

FLASK_ENV=production

SECRET_KEY=votre_cle_super_secrete

DATABASE_URL=postgresql://user:password@localhost/dbname

.gitignore

venv/

__pycache__/

*.pyc

.env

instance/

*.db

app.py (utiliser les variables d'environnement)

import os

from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)

app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY')

app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL')

Serveur WSGI : Gunicorn

En production, n'utilisez PAS le serveur de développement Flask !

pip install gunicorn

Lancer avec Gunicorn :

gunicorn -w 4 -b 0.0.0.0:8000 app:app

- -w 4 : 4 workers (processus)
- -b 0.0.0.0:8000 : Bind sur toutes les interfaces, port 8000
- app:app : module:application

Déploiement sur Heroku

1. Préparer l'application

Procfile (à la racine)

web: gunicorn app:app

runtime.txt (optionnel)

python-3.11.0

2. Installer Heroku CLI

Installation

https://devcenter.heroku.com/articles/heroku-cli

Se connecter

heroku login

3. Déployer

Créer une application

heroku create nom-de-votre-app

Ajouter PostgreSQL (gratuit)

heroku addons:create heroku-postgresql:mini

Configurer les variables

heroku config:set SECRET_KEY=votre_cle_secrete

Déployer

git push heroku main

Créer les tables

heroku run python

>>> from app import app, db

>>> with app.app_context():

>>> db.create_all()

Ouvrir l'application

heroku open

Déploiement sur PythonAnywhere

1. Créez un compte sur pythonanywhere.com
2. Ouvrez une console Bash
3. Clonez votre projet :
4. git clone https://github.com/votre-repo.gitcd votre-repo
5. Créez un environnement virtuel :
6. mkvirtualenv --python=/usr/bin/python3.10 mon-envpip install -r requirements.txt
7. Configurez l'application web dans l'onglet "Web"
8. Configurez le fichier WSGI

Déploiement avec Docker

Dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

```
COPY . .

EXPOSE 5000

CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "app:app"]
```

docker-compose.yml

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - DATABASE_URL=postgresql://user:pass@db:5432/mydb
    depends_on:
      - db

  db:
    image: postgres:15
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
      - POSTGRES_DB=mydb
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

```
volumes:
  postgres_data:
```

Commandes Docker :

```
# Build et lancer
docker-compose up --build

# En arrière-plan
docker-compose up -d

# Arrêter
docker-compose down
```

Bonnes Pratiques de Production

1. Configuration

```
class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL')
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class DevelopmentConfig(Config):
    DEBUG = True

class ProductionConfig(Config):
    DEBUG = False

config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig
}
```

```
app.config.from_object(config[os.environ.get('FLASK_ENV', 'production')])
```

2. Logging

```
import logging
from logging.handlers import RotatingFileHandler

if not app.debug:
    file_handler = RotatingFileHandler('app.log', maxBytes=10240, backupCount=10)
    file_handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]'
    ))
    file_handler.setLevel(logging.INFO)
    app.logger.addHandler(file_handler)
    app.logger.setLevel(logging.INFO)
    app.logger.info('Application startup')
```

3. Gestion d'Erreurs

```
@app.errorhandler(500)
def internal_error(error):
    db.session.rollback()
    app.logger.error(f'Server Error: {error}')
    return render_template('500.html'), 500

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

Exercice 10 : Déploiement Complet

Consigne : Déployez votre application de blog :

1. **Préparation** :

- Créez requirements.txt
- Configurez les variables d'environnement
- Créez .gitignore
- Utilisez PostgreSQL au lieu de SQLite

2. **Déployez sur Heroku ou PythonAnywhere**
3. **Configurez** :
   - HTTPS
   - Nom de domaine personnalisé (optionnel)
   - Logging
4. **Testez** :
   - Toutes les fonctionnalités
   - Performance
   - Sécurité
5. **Documentation** :
   - README.md avec instructions d'installation
   - Documentation de l'API (si applicable)

Conclusion

Félicitations ! Vous maîtrisez maintenant :

✅ Les bases de Flask et du routing
✅ Les templates avec Jinja2
✅ Les formulaires et validation
✅ Les bases de données avec SQLAlchemy
✅ L'authentification et les sessions
✅ Les fichiers statiques et le design
✅ La création d'APIs REST
✅ Le déploiement en production

Ressources pour Aller Plus Loin

Documentation officielle :

- Flask Documentation

- SQLAlchemy Documentation
- Jinja2 Documentation

Extensions Flask utiles :

- **Flask-Mail** : Envoi d'emails
- **Flask-Admin** : Interface d'administration
- **Flask-Migrate** : Migrations de base de données
- **Flask-Caching** : Mise en cache
- **Flask-SocketIO** : WebSockets en temps réel

Projets pratiques suggérés :

1. Blog complet avec commentaires et catégories
2. E-commerce simplifié
3. Réseau social minimaliste
4. API pour application mobile
5. Dashboard d'analyse de données

Bon développement web ! 🚀