

Cours Complet : Développement Web avec Flask

Table des matières

1. [Introduction au Développement Web](#)
 2. [Installation et Première Application Flask](#)
 3. [Routing et URLs](#)
 4. [Templates avec Jinja2](#)
 5. [Formulaires et Données POST](#)
 6. [Bases de Données avec SQLite](#)
 7. [Sessions et Authentification](#)
 8. [Fichiers Statiques \(CSS, JavaScript, Images\)](#)
 9. [API REST avec Flask](#)
 10. [Déploiement d'une Application Flask](#)
-

1. Introduction au Développement Web

Qu'est-ce que le Développement Web ?

Le développement web consiste à créer des sites et applications accessibles via un navigateur. Il se divise en deux parties :

Frontend (Client)

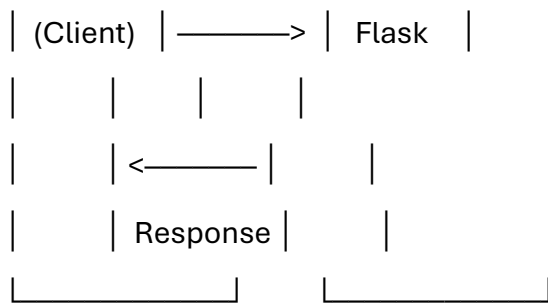
- Ce que l'utilisateur voit et avec quoi il interagit
- Technologies : HTML, CSS, JavaScript
- S'exécute dans le navigateur

Backend (Serveur)

- La logique de l'application, les bases de données
- Technologies : Python (Flask, Django), Node.js, PHP, etc.
- S'exécute sur le serveur

Architecture Client-Serveur





Qu'est-ce que Flask ?

Flask est un micro-framework web Python créé par Armin Ronacher en 2010.

Avantages :

- ☒ Léger et simple à apprendre
- ☒ Flexible : vous contrôlez tout
- ☒ Grande communauté et documentation
- ☒ Parfait pour les petits et moyens projets
- ☒ Extensions disponibles pour tout (bases de données, authentification, etc.)

Flask vs Django :

- **Flask** : Minimaliste, vous ajoutez ce dont vous avez besoin
- **Django** : "Batteries included", tout est intégré mais plus complexe

Le Protocole HTTP

HTTP (HyperText Transfer Protocol) est le langage du web.

Méthodes HTTP principales :

- **GET** : Récupérer des données (afficher une page)
- **POST** : Envoyer des données (soumettre un formulaire)
- **PUT** : Mettre à jour des données
- **DELETE** : Supprimer des données

Codes de statut HTTP :

- **200** : OK (succès)
- **404** : Not Found (page non trouvée)
- **500** : Internal Server Error (erreur serveur)
- **302** : Redirect (redirection)

Exercice 1 : Recherche et Compréhension

Consigne : Répondez aux questions suivantes :

1. Quelle est la différence entre le frontend et le backend ?
 2. Qu'est-ce qu'un framework web ?
 3. Citez 3 avantages de Flask
 4. Quelle méthode HTTP utiliseriez-vous pour :
 - Afficher une page d'accueil ?
 - Envoyer un formulaire de contact ?
 - Supprimer un article de blog ?
 5. Que signifie le code HTTP 404 ?
-

2. Installation et Première Application Flask

Installation de Flask

Étape 1 : Créer un Environnement Virtuel (Recommandé)

Un environnement virtuel isole les dépendances de votre projet.

Créer un environnement virtuel

```
python -m venv venv
```

Activer l'environnement (Windows)

```
venv\Scripts\activate
```

Activer l'environnement (Mac/Linux)

```
source venv/bin/activate
```

Étape 2 : Installer Flask

```
pip install flask
```

Votre Première Application Flask

Fichier : app.py

```
from flask import Flask
```

```
# Créer une instance de l'application Flask
app = Flask(__name__)

# Définir une route (URL)
@app.route('/')
def home():
    return "Bienvenue sur mon premier site Flask !"

@app.route('/about')
def about():
    return "À propos de nous"

# Lancer l'application
if __name__ == '__main__':
    app.run(debug=True)
```

Lancer l'application :

python app.py

Accéder à votre site :

- Ouvrez votre navigateur
- Allez sur : <http://127.0.0.1:5000/>

Explication du Code

```
app = Flask(__name__)

• Crée une instance de l'application Flask
• __name__ aide Flask à trouver les fichiers

@app.route('/')

def home():

    return "Bienvenue..."
```




- `@app.route('/') :` Décorateur qui lie l'URL / à la fonction
- La fonction retourne ce qui sera affiché dans le navigateur

`app.run(debug=True)`

- Lance le serveur de développement
- `debug=True` : Active le mode debug (redémarrage auto, messages d'erreur détaillés)

Mode Debug

Avantages du mode debug :

-  Redémarrage automatique quand vous modifiez le code
-  Messages d'erreur détaillés dans le navigateur
-  Debugger interactif

 **IMPORTANT : Ne jamais utiliser `debug=True` en production !**

Structure de Projet Recommandée

```
mon_projet/
|
|— venv/          # Environnement virtuel
|— app.py         # Application principale
|— templates/     # Fichiers HTML
|— static/        # CSS, JS, images
|  |— css/
|  |— js/
|  └─ img/
└─ requirements.txt # Liste des dépendances
```

Fichier requirements.txt

`Flask==3.0.0`

Pour installer les dépendances plus tard :

`pip install -r requirements.txt`

Exercice 2 : Première Application

Consigne : Créez une application Flask avec :

1. Une page d'accueil (/) qui affiche "Bienvenue sur mon portfolio"
 2. Une page À propos (/about) qui affiche votre présentation
 3. Une page Compétences (/skills) qui liste vos compétences
 4. Une page Contact (/contact) qui affiche vos informations de contact
 5. Testez chaque route dans le navigateur
 6. Activez le mode debug et observez ce qui se passe quand vous modifiez le code
-

3. Routing et URLs

Routes Basiques

```
from flask import Flask
```

```
app = Flask(__name__)
```

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

```
def home():
```

```
    return "Page d'accueil"
```

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

```
def blog():
```

```
    return "Page du blog"
```

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

```
def contact():
```

```
    return "Page de contact"
```

Routes avec Variables

```
@app.route('/user/<username>')
```

```
def show_user(username):
```

```
return f"Profil de l'utilisateur : {username}"
```

```
@app.route('/post/<int:post_id>')
```

```
def show_post(post_id):
```

```
    return f"Article numéro : {post_id}"
```

```
@app.route('/product/<float:price>')
```

```
def show_price(price):
```

```
    return f"Prix du produit : {price}€"
```

Types de convertisseurs :

- <string:name> : Chaîne de caractères (défaut)
- <int:id> : Entier
- <float:price> : Nombre décimal
- <path:subpath> : Chaîne avec slashes

Exemples d'URLs dynamiques

```
# URL : /article/42
```

```
@app.route('/article/<int:article_id>')
```

```
def article(article_id):
```

```
    return f"Vous lisez l'article {article_id}"
```

```
# URL : /user/alice/posts
```

```
@app.route('/user/<username>/posts')
```

```
def user_posts(username):
```

```
    return f"Articles de {username}"
```

```
# URL : /files/documents/rapport.pdf
```

```
@app.route('/files/<path:filename>')
```

```
def download_file(filename):
```

```
return f"Téléchargement de : {filename}"
```

Méthodes HTTP

Par défaut, une route n'accepte que GET. Pour accepter d'autres méthodes :

```
from flask import request
```

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

```
def login():
```

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

```
        return "Traitement du formulaire de connexion"
```

```
    else:
```

```
        return "Affichage du formulaire de connexion"
```

```
@app.route('/api/data', methods=['GET', 'POST', 'PUT', 'DELETE'])
```

```
def api_data():
```

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

```
        return "Récupération des données"
```

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

```
        return "Création de données"
```

```
    elif request.method == 'PUT':
```

```
        return "Mise à jour de données"
```

```
    elif request.method == 'DELETE':
```

```
        return "Suppression de données"
```

Redirection et Erreurs

```
from flask import redirect, url_for, abort
```

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

```
def home():
```

```
    return "Accueil"
```



```

@app.route('/admin')
def admin():
    # Rediriger vers la page d'accueil
    return redirect(url_for('home'))

@app.route('/user/<int:user_id>')
def show_user(user_id):
    if user_id <= 0:
        abort(404) # Déclencher une erreur 404
    return f"Utilisateur {user_id}"

# Gérer les erreurs 404
@app.errorhandler(404)
def page_not_found(error):
    return "Cette page n'existe pas !", 404

# Gérer les erreurs 500
@app.errorhandler(500)
def internal_error(error):
    return "Erreur interne du serveur", 500

```

url_for() : Générer des URLs

```

from flask import url_for

```

```

@app.route('/')
def home():
    return "Accueil"

```

```

@app.route('/user/<username>')
def profile(username):
    return f"Profil de {username}"

@app.route('/test')
def test():
    # Générer l'URL de la fonction home
    home_url = url_for('home') # '/'

    # Générer l'URL avec paramètres
    user_url = url_for('profile', username='alice') # '/user/alice'

    return f"URL Accueil: {home_url}<br>URL Profil: {user_url}"

```

Avantages de url_for() :

- Si vous changez l'URL, le code continue de fonctionner
- Plus sûr que d'écrire les URLs en dur
- Gère automatiquement les paramètres

Exercice 3 : Routing Avancé

Consigne : Créez une application de blog avec :

1. Une page d'accueil (/)
2. Une page liste des articles (/articles)
3. Une page pour un article spécifique (/articles/<int:id>)
4. Une page auteur (/author/<name>)
5. Une page catégorie (/category/<category_name>)
6. Une redirection de /home vers /
7. Une page 404 personnalisée
8. Une route /admin qui redirige vers /login
9. Utilisez url_for() pour générer des liens entre les pages

10. Affichez les informations de la requête (méthode HTTP, URL, etc.)

4. Templates avec Jinja2

Qu'est-ce que Jinja2 ?

Jinja2 est le moteur de templates de Flask. Il permet de :

- Séparer la logique Python du HTML
- Réutiliser du code HTML
- Créer des pages dynamiques

Structure des Templates

mon_projet/

├─ app.py

└─ templates/

 ├─ base.html

 ├─ home.html

 └─ about.html

Premier Template

Fichier : templates/home.html

```
<!DOCTYPE html>
```

```
<html lang="fr">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <title>Accueil</title>
```

```
</head>
```

```
<body>
```

```
    <h1>Bienvenue sur mon site !</h1>
```

```
    <p>Ceci est une page HTML.</p>
```

```
</body>
```

```
</html>
```

Fichier : app.py

```
from flask import Flask, render_template
```

```
app = Flask(__name__)
```

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

```
def home():
```

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

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

Passer des Variables aux Templates**app.py**

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

```
def home():
```

```
    nom = "Alice"
```

```
    age = 25
```

```
    return render_template('home.html', nom=nom, age=age)
```

```
@app.route('/user/<username>')
```

```
def user_profile(username):
```

```
    user_data = {
```

```
        'username': username,
```

```
        'email': f'{username}@example.com',
```

```
        'posts': 42
```

```
    }
```

```
    return render_template('profile.html', user=user_data)
```

templates/home.html

```
<!DOCTYPE html>

<html lang="fr">

<head>

  <title>Accueil</title>

</head>

<body>

  <h1>Bonjour {{ nom }} !</h1>

  <p>Vous avez {{ age }} ans.</p>

</body>

</html>
```

Syntaxe Jinja2

Variables

```
{{ variable }}

{{ user.name }}

{{ list[0] }}

{{ dict['key'] }}
```

Conditions

```
{% if age >= 18 %}

  <p>Vous êtes majeur.</p>

{% else %}

  <p>Vous êtes mineur.</p>

{% endif %}


{% if user %}

  <p>Bienvenue {{ user.name }}</p>

{% else %}

  <p>Veuillez vous connecter</p>

{% endif %}
```

Boucles

```
<ul>
```

```
{% for item in items %}
```

```
    <li>{{ item }}</li>
```

```
{% endfor %}
```

```
</ul>
```

```
<!-- Avec index -->
```

```
{% for i, item in enumerate(items) %}
```

```
    <p>{{ i + 1 }}. {{ item }}</p>
```

```
{% endfor %}
```

```
<!-- Boucle vide -->
```

```
{% for post in posts %}
```

```
    <p>{{ post.title }}</p>
```

```
{% else %}
```

```
    <p>Aucun article disponible.</p>
```

```
{% endfor %}
```

Filtres

Les filtres modifient les variables :

```
{{ name|upper }}      <!-- ALICE -->
```

```
{{ name|lower }}      <!-- alice -->
```

```
{{ name|capitalize }} <!-- Alice -->
```

```
{{ name|title }}      <!-- Alice Dupont -->
```

```
{{ text|truncate(20) }} <!-- Tronque à 20 caractères -->
```

```
{{ number|round(2) }}  <!-- Arrondit à 2 décimales -->
```

```
{{ list|length }}      <!-- Nombre d'éléments -->
```

```
{{ date|date('dd/mm/yyyy') }} <!-- Formate une date -->
```

```
{{ "<p>HTML</p>"|safe }} <!-- N'échappe pas le HTML -->
```

Template d'Héritage (Layouts)

templates/base.html (Template parent)

```
<!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>
```

```
</head>
```

```
<body>
```

```
    <header>
```

```
        <nav>
```

```
            <a href="/">Accueil</a>
```

```
            <a href="/about">À propos</a>
```

```
            <a href="/blog">Blog</a>
```

```
        </nav>
```

```
    </header>
```

```
    <main>
```

```
        {% block content %}{% endblock %}
```

```
    </main>
```

```
    <footer>
```

```
        <p>&copy; 2024 Mon Site Web</p>
```

```
    </footer>
```

```
</body>
```

```
</html>
```

templates/home.html (Template enfant)

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

```
{% block title %}Accueil - Mon Site{% endblock %}
```

```
{% block content %}
```

```
    <h1>Bienvenue sur mon site !</h1>
```

```
    <p>Ceci est la page d'accueil.</p>
```

```
{% endblock %}
```

templates/blog.html

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

```
{% block title %}Blog{% endblock %}
```

```
{% block content %}
```

```
    <h1>Articles du Blog</h1>
```

```
    {% for post in posts %}
```

```
        <article>
```

```
            <h2>{{ post.title }}</h2>
```

```
            <p>{{ post.content|truncate(100) }}</p>
```

```
            <a href="/post/{{ post.id }}">Lire la suite</a>
```

```
        </article>
```

```
    {% endfor %}
```

```
{% endblock %}
```

Inclure des Fragments (Include)

templates/navbar.html


```
<nav>

  <a href="/">Accueil</a>

  <a href="/blog">Blog</a>

  <a href="/contact">Contact</a>

</nav>
```

templates/base.html

```
<!DOCTYPE html>

<html>

<head>

  <title>Mon Site</title>

</head>

<body>

  {% include 'navbar.html' %}


  {% block content %}{% endblock %}

</body>

</html>
```

Macros (Fonctions Réutilisables)

templates/macros.html

```
{% macro render_post(post) %}

  <article class="post">

    <h2>{{ post.title }}</h2>

    <p class="date">{{ post.date }}</p>

    <p>{{ post.content }}</p>

  </article>

{% endmacro %}


{% macro render_button(text, url, style="primary") %}
```

```

<a href="{{ url }}" class="btn btn-{{ style }}">
    {{ text }}
</a>

{% endmacro %}

templates/blog.html

{% from 'macros.html' import render_post, render_button %}

{% for post in posts %}
    {{ render_post(post) }}
{% endfor %}

{{ render_button("Lire plus", "/blog", "primary") }}

```

Exemple Complet

app.py

```

from flask import Flask, render_template
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html',
                           title="Accueil",
                           current_year=datetime.now().year)

@app.route('/blog')
def blog():
    posts = [

```

```
{
    'id': 1,
    'title': 'Premier article',
    'content': 'Contenu du premier article...',
    'author': 'Alice',
    'date': '01/01/2024'
},
{
    'id': 2,
    'title': 'Deuxième article',
    'content': 'Contenu du deuxième article...',
    'author': 'Bob',
    'date': '02/01/2024'
}
]

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

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    # En réalité, on récupérerait ceci d'une base de données
    post = {
        'id': post_id,
        'title': f'Article {post_id}',
        'content': 'Contenu complet de l'article...',
        'author': 'Alice',
        'date': '01/01/2024'
    }
    return render_template('post.html', post=post)
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

Exercice 4 : Templates et Jinja2

Consigne : Créez un site de portfolio avec templates :

1. Créez un template de base (base.html) avec :
 - Un header avec navigation
 - Un bloc pour le contenu
 - Un footer
2. Créez une page d'accueil qui hérite de base.html
3. Créez une page "Projets" qui affiche une liste de projets :
 - Chaque projet a : titre, description, technologies, lien
 - Utilisez une boucle pour afficher les projets
4. Créez une page détail de projet (/project/<int:id>)
5. Créez une macro pour afficher une carte de projet
6. Utilisez des filtres Jinja2 (uppercase, truncate, etc.)
7. Affichez la date actuelle dans le footer
8. Ajoutez des conditions : si aucun projet, afficher un message

5. Formulaires et Données POST

Formulaire HTML de Base

templates/contact.html

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

```
{% block content %}
```

```
<h1>Contactez-nous</h1>
```

```
<form method="POST" action="/contact">
```

```
<div>

    <label for="name">Nom :</label>

    <input type="text" id="name" name="name" required>

</div>
```

```
<div>

    <label for="email">Email :</label>

    <input type="email" id="email" name="email" required>

</div>
```

```
<div>

    <label for="message">Message :</label>

    <textarea id="message" name="message" rows="5" required></textarea>

</div>
```

```
<button type="submit">Envoyer</button>
```

```
</form>
```

```
{% endblock %}
```

Traiter les Données du Formulaire

app.py

```
from flask import Flask, render_template, request, redirect, url_for, flash
```

```
app = Flask(__name__)
```

```
app.secret_key = 'votre_cle_secrete_ici' # Nécessaire pour flash()
```

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

```
def contact():
```

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

```
# Récupérer les données du formulaire

name = request.form['name']
email = request.form['email']
message = request.form['message']


# Traiter les données (sauvegarder, envoyer email, etc.)

print(f"Nouveau message de {name} ({email}): {message}")


# Message flash de succès

flash('Votre message a été envoyé avec succès !', 'success')


return redirect(url_for('contact'))


# GET : Afficher le formulaire

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

Messages Flash

Les messages flash permettent d'afficher des notifications temporaires.

app.py

```
from flask import flash


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

    if request.method == 'POST':

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


        if username == 'admin' and password == 'secret':

            flash('Connexion réussie !', 'success')
```

```
        return redirect(url_for('dashboard'))

    else:

        flash('Identifiants incorrects', 'error')
```

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

templates/base.html

```
<!DOCTYPE html>

<html>

<head>

    <title>Mon Site</title>

    <style>

        .flash-success { background: #d4edda; color: #155724; padding: 10px; }

        .flash-error { background: #f8d7da; color: #721c24; padding: 10px; }

    </style>

</head>

<body>

    <!-- Afficher les messages flash -->

    {% with messages = get_flashed_messages(with_categories=true) %}

        {% if messages %}

            {% for category, message in messages %}

                <div class="flash-{{ category }}">

                    {{ message }}

                </div>

            {% endfor %}

        {% endif %}

    {% endwith %}


    {% block content %}{% endblock %}
```

```
</body>
```

```
</html>
```

Validation de Formulaires

Validation basique

```
from flask import request, flash
```

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

```
def register():
```

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

```
        username = request.form.get('username', "").strip()
```

```
        email = request.form.get('email', "").strip()
```

```
        password = request.form.get('password', "")
```

```
    # Validation
```

```
    errors = []
```

```
    if len(username) < 3:
```

```
        errors.append('Le nom d\'utilisateur doit contenir au moins 3 caractères')
```

```
    if '@' not in email:
```

```
        errors.append('Email invalide')
```

```
    if len(password) < 8:
```

```
        errors.append('Le mot de passe doit contenir au moins 8 caractères')
```

```
    if errors:
```

```
        for error in errors:
```

```
            flash(error, 'error')
```



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

```
# Si tout est OK
```

```
flash('Inscription réussie !', 'success')
```

```
return redirect(url_for('login'))
```

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

Flask-WTF (Extension pour Formulaires)

Installation :

```
pip install flask-wtf
```

app.py

```
from flask import Flask, render_template, redirect, url_for, flash
```

```
from flask_wtf import FlaskForm
```

```
from wtforms import StringField, TextAreaField, SubmitField, EmailField
```

```
from wtforms.validators import DataRequired, Email, Length
```

```
app = Flask(__name__)
```

```
app.secret_key = 'votre_cle_secrete'
```

```
# Définir un formulaire
```

```
class ContactForm(FlaskForm):
```

```
    name = StringField('Nom',
```

```
        validators=[DataRequired(), Length(min=2, max=50)])
```

```
    email = EmailField('Email',
```

```
        validators=[DataRequired(), Email()])
```

```
    message = TextAreaField('Message',
```

```
        validators=[DataRequired(), Length(min=10)])
```

```
    submit = SubmitField('Envoyer')
```

```
@app.route('/contact', methods=['GET', 'POST'])
def contact():
    form = ContactForm()

    if form.validate_on_submit():
        # Les données sont valides
        name = form.name.data
        email = form.email.data
        message = form.message.data

        # Traiter les données
        print(f"Message de {name} ({email}): {message}")

        flash('Message envoyé avec succès !', 'success')
        return redirect(url_for('contact'))

    return render_template('contact.html', form=form)
```

templates/contact.html

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

{% block content %}
<h1>Contactez-nous</h1>

<form method="POST" novalidate>
    {{ form.hidden_tag() }}

    <div>
```

```
{{ form.name.label }}  
  
{{ form.name(size=32) }}  
  
{% if form.name.errors %}  
    <ul class="errors">  
        {% for error in form.name.errors %}  
            <li>{{ error }}</li>  
        {% endfor %}  
    </ul>  
  
{% endif %}  
</div>
```

```
<div>  
  
    {{ form.email.label }}  
  
    {{ form.email(size=32) }}  
  
    {% if form.email.errors %}  
        <ul class="errors">  
            {% for error in form.email.errors %}  
                <li>{{ error }}</li>  
            {% endfor %}  
        </ul>  
  
    {% endif %}  
</div>
```

```
<div>  
  
    {{ form.message.label }}  
  
    {{ form.message(rows=5) }}  
  
    {% if form.message.errors %}  
        <ul class="errors">
```

```

        {% for error in form.message.errors %}

            <li>{{ error }}</li>

        {% endfor %}

    </ul>

{% endif %}

</div>


<div>

    {{ form.submit() }}

</div>

</form>

{% endblock %}

```

Upload de Fichiers

templates/upload.html

```

<form method="POST" enctype="multipart/form-data">

    <input type="file" name="file" required>

    <button type="submit">Uploader</button>

</form>

```

app.py

```

import os

from flask import request

from werkzeug.utils import secure_filename


UPLOAD_FOLDER = 'uploads'

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'pdf'}


app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 # 16 MB max

```

```

def allowed_file(filename):

    return '.' in filename and \

        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/upload', methods=['GET', 'POST'])
def upload_file():

    if request.method == 'POST':

        # Vérifier si un fichier est présent

        if 'file' not in request.files:

            flash('Aucun fichier sélectionné', 'error')

            return redirect(request.url)

        file = request.files['file']

        # Vérifier si un fichier est sélectionné

        if file.filename == '':

            flash('Aucun fichier sélectionné', 'error')

            return redirect(request.url)

        # Vérifier l'extension

        if file and allowed_file(file.filename):

            filename = secure_filename(file.filename)

            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)

            file.save(filepath)

            flash(f'Fichier {filename} uploadé avec succès !', 'success')

            return redirect(url_for('upload_file'))

```

else:

flash('Type de fichier non autorisé', 'error')

return render_template('upload.html')

Exercice 5 : Formulaires Complets

Consigne : Créez une application de gestion de tâches (To-Do List) :

1. **Page d'accueil :** Affiche toutes les tâches
2. **Formulaire d'ajout :**
 - Titre (obligatoire, 3-100 caractères)
 - Description (optionnel)
 - Priorité (Basse/Moyenne/Haute)
 - Date limite (optionnel)
3. **Validation :** Vérifiez que le titre n'est pas vide
4. **Messages flash :** Confirmez l'ajout/suppression
5. **Fonctionnalités :**
 - Ajouter une tâche
 - Marquer comme terminée
 - Supprimer une tâche
 - Modifier une tâche
6. Stockez les tâches dans une liste (pour l'instant, pas de BDD)
7. Utilisez Flask-WTF pour le formulaire
8. Ajoutez du CSS pour rendre l'interface agréable

6. Bases de Données avec SQLite

Introduction aux Bases de Données

Flask peut utiliser plusieurs bases de données :

- **SQLite :** Simple, fichier local (idéal pour débiter)
- **PostgreSQL :** Puissant, production

- **MySQL** : Populaire, production
- **MongoDB** : Base NoSQL

Nous allons utiliser **SQLite** avec **Flask-SQLAlchemy**.

Installation

```
pip install flask-sqlalchemy
```

Configuration de la Base de Données

app.py

```
from flask import Flask
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
from datetime import datetime
```

```
app = Flask(__name__)
```

```
# Configuration de la base de données
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
```

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
# Initialiser SQLAlchemy
```

```
db = SQLAlchemy(app)
```

Créer un Modèle (Table)

```
class User(db.Model):
```

```
    """Modèle pour les utilisateurs"""
```

```
    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)
```

```
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
    def __repr__(self):
```

```
return f'<User {self.username}>'
```

```
class Post(db.Model):
```

```
    """Modèle pour les articles de blog"""
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    title = db.Column(db.String(200), nullable=False)
```

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

```
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

```
    # Relation avec User
```

```
    author = db.relationship('User', backref=db.backref('posts', lazy=True))
```

```
    def __repr__(self):
```

```
        return f'<Post {self.title}>'
```

Types de Colonnes

db.Integer # Entier

db.String(length) # Chaîne de caractères (longueur max)

db.Text # Texte long

db.DateTime # Date et heure

db.Float # Nombre décimal

db.Boolean # Booléen

db.PickleType # Objet Python sérialisé

Contraintes

primary_key=True # Clé primaire

unique=True # Valeur unique

nullable=False # Non-null (obligatoire)

default=value # Valeur par défaut


```
index=True      # Créer un index
```

Créer les Tables

Dans le terminal Python ou un script :

```
from app import app, db
```

```
# Créer toutes les tables
```

```
with app.app_context():
```

```
    db.create_all()
```

```
    print("Tables créées !")
```

Ou dans app.py :

```
if __name__ == '__main__':
```

```
    with app.app_context():
```

```
        db.create_all()
```

```
    app.run(debug=True)
```

CRUD : Create (Créer)

```
@app.route('/add_user', methods=['POST'])
```

```
def add_user():
```

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

```
    email = request.form['email']
```

```
# Créer un nouvel utilisateur
```

```
new_user = User(username=username, email=email)
```

```
try:
```

```
    # Ajouter à la session
```

```
    db.session.add(new_user)
```

```
    # Enregistrer dans la base de données
```

```
    db.session.commit()
```

```

        flash('Utilisateur créé avec succès !', 'success')

except Exception as e:

    db.session.rollback()

    flash(f'Erreur : {str(e)}', 'error')


return redirect(url_for('home'))

CRUD : Read (Lire)

# Récupérer tous les utilisateurs

@app.route('/users')

def list_users():

    users = User.query.all()

    return render_template('users.html', users=users)


# Récupérer un utilisateur par ID

@app.route('/user/<int:user_id>')

def show_user(user_id):

    user = User.query.get_or_404(user_id)

    return render_template('user.html', user=user)


# Recherche avec filtre

@app.route('/search')

def search():

    keyword = request.args.get('q', '')

    users = User.query.filter(User.username.like(f'%{keyword}%')).all()

    return render_template('search.html', users=users)


# Premiers résultats

```

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

def recent_posts():

    posts = Post.query.order_by(Post.created_at.desc()).limit(5).all()

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

CRUD : Update (Mettre à Jour)

```
@app.route('/user/<int:user_id>/edit', methods=['GET', 'POST'])
```

```
def edit_user(user_id):

    user = User.query.get_or_404(user_id)

    if request.method == 'POST':

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

        user.email = request.form['email']

        try:

            db.session.commit()

            flash('Utilisateur mis à jour !', 'success')

            return redirect(url_for('show_user', user_id=user.id))

        except Exception as e:

            db.session.rollback()

            flash(f'Erreur : {str(e)}', 'error')

    return render_template('edit_user.html', user=user)
```

CRUD : Delete (Supprimer)

```
@app.route('/user/<int:user_id>/delete', methods=['POST'])
```

```
def delete_user(user_id):

    user = User.query.get_or_404(user_id)

    try:
```

```

    db.session.delete(user)

    db.session.commit()

    flash('Utilisateur supprimé', 'success')

except Exception as e:

    db.session.rollback()

    flash(f'Erreur : {str(e)}', 'error')


return redirect(url_for('list_users'))

```

Relations entre Tables

One-to-Many (Un à Plusieurs)

```

class Author(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    name = db.Column(db.String(100), nullable=False)


# Relation : un auteur a plusieurs livres

books = db.relationship('Book', backref='author', lazy=True)


class Book(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    title = db.Column(db.String(200), nullable=False)

    author_id = db.Column(db.Integer, db.ForeignKey('author.id'), nullable=False)


# Utilisation

author = Author.query.first()

for book in author.books:

    print(book.title)


book = Book.query.first()

```

```
print(f"Écrit par : {book.author.name}")
```

Many-to-Many (Plusieurs à Plusieurs)

```
# Table d'association
```

```
tags = db.Table('post_tags',
    db.Column('post_id', db.Integer, db.ForeignKey('post.id'), primary_key=True),
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'), primary_key=True)
)
```

```
class Post(db.Model):
```

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

```
# Relation many-to-many
```

```
tags = db.relationship('Tag', secondary=tags, lazy='subquery',
    backref=db.backref('posts', lazy=True))
```

```
class Tag(db.Model):
```

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

```
# Utilisation
```

```
post = Post.query.first()
```

```
for tag in post.tags:
```

```
    print(tag.name)
```

Requêtes Avancées

```
# Filtrage
```

```
User.query.filter_by(username='alice').first()
```

```
User.query.filter(User.email.endswith('@gmail.com')).all()
```

Opérateurs de comparaison

```
Post.query.filter(Post.created_at > datetime(2024, 1, 1)).all()
```

```
Post.query.filter(Post.title.like('%Python%')).all()
```

Combinaison de filtres (AND)

```
User.query.filter(User.username == 'alice', User.email.endswith('.com')).first()
```

OR

```
from sqlalchemy import or_
```

```
User.query.filter(or_(User.username == 'alice', User.username == 'bob')).all()
```

IN

```
User.query.filter(User.id.in_([1, 2, 3])).all()
```

Tri

```
Post.query.order_by(Post.created_at.desc()).all()
```

```
Post.query.order_by(Post.title.asc()).all()
```

Limite et offset (pagination)

```
Post.query.limit(10).all()
```

```
Post.query.offset(10).limit(10).all()
```

Compter

```
User.query.count()
```

```
Post.query.filter_by(user_id=1).count()
```

Exemple Complet : Blog

app.py

```
from flask import Flask, render_template, request, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
```

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.secret_key = 'secret_key_here'
```

```
db = SQLAlchemy(app)
```

```
# Modèles
```

```
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    content = db.Column(db.Text, nullable=False)
    author = db.Column(db.String(100), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
# Routes
```

```
@app.route('/')
def home():
    posts = Post.query.order_by(Post.created_at.desc()).all()
    return render_template('home.html', posts=posts)
```

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    post = Post.query.get_or_404(post_id)
```

```
return render_template('post.html', post=post)
```

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

```
def new_post():
```

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

```
        title = request.form['title']
```

```
        content = request.form['content']
```

```
        author = request.form['author']
```

```
        new_post = Post(title=title, content=content, author=author)
```

```
        db.session.add(new_post)
```

```
        db.session.commit()
```

```
        flash('Article publié !', 'success')
```

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

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

```
@app.route('/post/<int:post_id>/edit', methods=['GET', 'POST'])
```

```
def edit_post(post_id):
```

```
    post = Post.query.get_or_404(post_id)
```

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

```
        post.title = request.form['title']
```

```
        post.content = request.form['content']
```

```
        post.author = request.form['author']
```

```
        db.session.commit()
```



```

    flash('Article mis à jour !', 'success')

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

return render_template('edit_post.html', post=post)

@app.route('/post/<int:post_id>/delete', methods=['POST'])
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    db.session.delete(post)
    db.session.commit()

    flash('Article supprimé', 'success')
    return redirect(url_for('home'))

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)

```

Exercice 6 : Application de Blog Complète

Consigne : Créez une application de blog avec base de données :

1. Modèles :

- User (id, username, email, password_hash, created_at)
- Post (id, title, content, author_id, created_at)
- Comment (id, content, post_id, user_id, created_at)

2. Fonctionnalités :

- Créer/lire/modifier/supprimer des articles
- Ajouter des commentaires aux articles
- Lister tous les articles d'un auteur

- Rechercher des articles par titre
- Afficher les 5 derniers articles

3. Pages :

- Accueil : liste des articles
- Détail d'un article avec ses commentaires
- Formulaire de nouvel article
- Profil d'utilisateur avec ses articles

4. Utilisez des relations entre tables

5. Ajoutez de la pagination (10 articles par page)

(Le cours continue dans la partie suivante...)