



# Flask

web development,  
one drop at a time

# Flask

## Micro framework de développement web

**SIMPLON**  
.CO

Présenté par Lamia & Abire

- 
- **Présentation de Flask**
  - **Création d'une application**

# Présentation de Flask

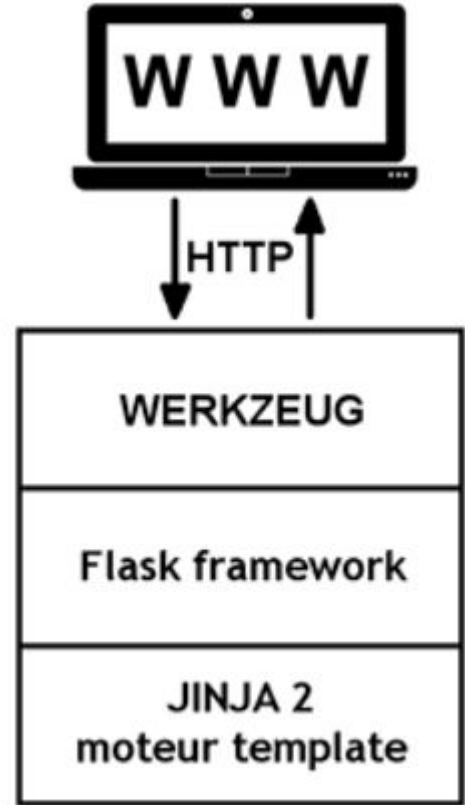
---

- Flask est un micro-framework web écrit en Python.
- Il vous permet de concevoir une application web solide de manière professionnelle.
- Flask s'est imposé comme un des frameworks les plus importants de la communauté Python(puissant et très léger).

# Structure du micro framework Flask:

***Flask est livré avec le strict minimum, à savoir :***

- un moteur de template (Jinja 2)
- un serveur web de développement (Werkzeug)
- un système de distribution de requête compatible REST (dit RESTful)
- un support de débogage intégré au serveur web
- un micro framework doté d'une très grande flexibilité
- une très bonne documentation



# Anatomie d'un projet Flask

L'élément le plus important d'un projet Flask est la structure des répertoires utilisée pour stocker les différents éléments du projet..

Flask, n'impose pas de structure, mais Suivre certaines conventions sera très utile

- les **feuilles de style, scripts, images** et autres éléments qui ne seront jamais générés dynamiquement doivent être dans le dossier static,
- les **fichiers HTML** doivent être dans le dossier templates,

```
mon-projet/  
└─ app  
    ├── __init__.py  
    ├── static  
    │   ├── website.css  
    │   ├── datagrid.css  
    │   ├── ico-new.png  
    │   ├── ico-top.png  
    │   └── logo.png  
    ├── templates  
    │   ├── index.html  
    │   ├── base.html  
    │   └── entries.html  
    ├── config.py  
    ├── views.py  
    └── ...
```

# Création d'une application de base :

une page web qui contient :

- un tableau qui liste les movies de la table “movies”
- un formulaire pour ajouter un film et l’insérer dans la BDD.

## Create a New movies

Title

Genre

Submit

## Movies List

TITRES	Genre
test	test
Taika Waititi (2020)	Comédie , Drame, Guerre
Andrew Dice Clay: Dice Rules (1991)	Comedy
Bungo Stray Dogs: Dead Apple (2018)	Action Animation
Flint (2017)	Drama
No Game No Life: Zero (2017)	Animation Comedy Fantasy
Black Butler: Book of the Atlantic (2017)	Action Animation Comedy Fantasy
Jon Stewart Has Left the Building (2015)	Documentary
Love Live! The School Idol Movie (2015)	Animation
Silver Spoon (2014)	Comedy Drama

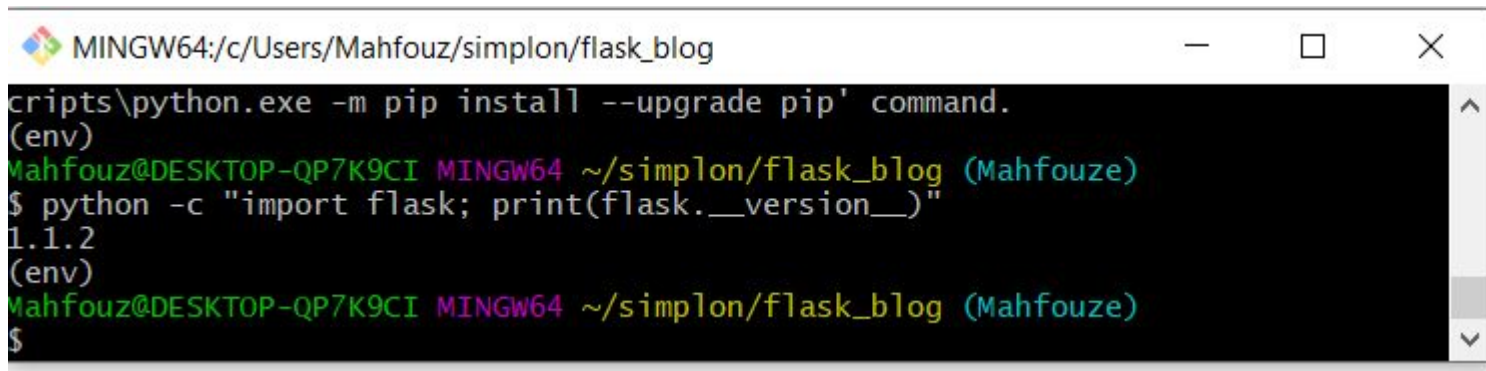
ADD movies

***la boîte à outils Bootstrap est utilisée pour styliser notre application***

# Installation de Flask

---

- création d'un répertoire Démo\_Flask
- création et activation d'un environnement virtuel(env)
- installer Flask: `pip install flask`
- confirmer l'installation: `python -c "import flask; print(flask.__version__)"`



```
MINGW64:/c/Users/Mahfouz/simplon/flask_blog
Scripts\python.exe -m pip install --upgrade pip' command.
(env)
Mahfouz@DESKTOP-QP7K9CI MINGW64 ~/simplon/flask_blog (Mahfouze)
$ python -c "import flask; print(flask.__version__)"
1.1.2
(env)
Mahfouz@DESKTOP-QP7K9CI MINGW64 ~/simplon/flask_blog (Mahfouze)
$
```

# Architecture du projet :

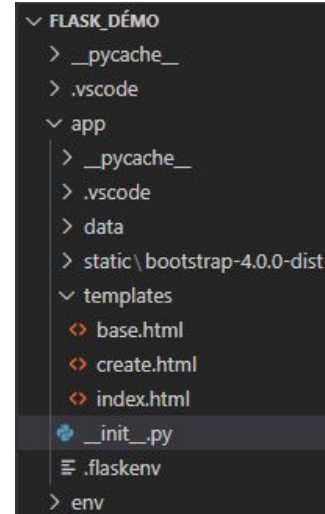
- dans le fichier `__init__.py` : on importe d'abord l'objet `Flask` du paquet `flask`. on l'utilise ensuite pour créer une instance d'application `Flask` avec le nom `app`. ensuite passer la variable spéciale `__name__` qui contient le nom du module Python actuel

```
from flask import Flask, render_template, request, url_for, flash, redirect
import sqlite3

def create_app():
    app = Flask(__name__)
```

- Une fois l'instance de l'application est créée, on utilise `@app.route` pour traiter les demandes web entrantes et envoyer des réponses à l'utilisateur. `@app.route` est un décorateur qui transforme une fonction Python ordinaire en une *fonction d'affichage* Flask,

```
@app.route('/add')
def create():
    return render_template('create.html')
```



The screenshot shows a file explorer for a project named 'FLASK DÉMO'. The structure is as follows:

- FLASK DÉMO
  - > \_\_pycache\_\_
  - > .vscode
  - app
    - > \_\_pycache\_\_
    - > .vscode
    - > data
    - > static\bootstrap-4.0.0-dist
    - templates
      - <> base.html
      - <> create.html
      - <> index.html
    - \_\_init\_\_.py (selected)
    - .flaskenv
  - > env

La fonction `render_template('template')` permet l'utilisation du moteur de modèle Jinja. et Cela facilitera la gestion du HTML



# Exécuter une application Flask

Pour exécuter votre application web, on doit d'abord indiquer à Flask où trouver l'application (le fichier `__init__.py` dans votre cas) avec la variable d'environnement `FLASK_APP`: `export FLASK_APP=__init__.py`

Ensuite, exécutez-la en mode développement avec la variable d'environnement `FLASK_ENV`:

`export FLASK_ENV=development` ( le mode développement permet d'activer le débogueur)

Pour terminer, exécutez l'application en utilisant la commande `flask run`

Ps : dans notre cas pour éviter de retaper les 3 commande pour chaque exécution, on a sauvegardé les deux variable `FLASK_APP` et `FLASK_ENV` dans un fichier `.flaskenv`, mais avant de le faire il faut installer `DOTenv` comme suit :

```
$ pip install -U python-dotenv
Collecting python-dotenv
  Using cached python_dotenv-0.15.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-0.15.0
(env)
```

# Affichage des movies

- Pour afficher les movies dans notre page d'accueil on a ajouter une fonction `index()` pour se connecter à la bdd sqlite3 et récupérer les movies ensuite on appelle le template `'index.html'` pour les afficher.

```
@app.route('/')
def index():
    conn = get_db_connection()
    movies = conn.execute('select * from movies order by movieId DESC limit 10 ').fetchall()
    conn.close()

    return render_template('index.html', movies=movies)

def get_db_connection():
    conn = sqlite3.connect('data/moviesdb.db')
    conn.row_factory = sqlite3.Row
    return conn
```

# Ajouter un nouveau film:

Pour permettre aux utilisateurs d'ajouter un nouveau film dans movies, on a défini la fonction

```
create_movies():
```

`request.form` donne accès aux données saisies par l'utilisateur.

```
@app.route('/create', methods=('GET', 'POST'))
def create_movies():
    if request.method == 'POST':
        title = request.form['title']
        genre = request.form['genre']

        if not title:
            flash('Title is required!')
        else:
            conn = get_db_connection()
            conn.execute('INSERT INTO movies (title, genres) values (?, ?)',
                        (title, genre))
            conn.commit()
            conn.close()
            return redirect(url_for('index'))

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

# Ajouter une nouvelle film:

Si le titre est fourni, vous ouvrez une connexion avec la fonction `conn = get_db_connection()` et insérez le titre et le genre que vous avez introduit dans le formulaire.

Vous validez ensuite les modifications dans la base de données et coupez la connexion.

On redirige l'utilisateur vers la page d'index en utilisant la fonction `redirect()` en lui passant l'URL générée par la fonction `url_for()` avec la valeur `'index'` en argument.

```
@app.route('/create', methods=('GET', 'POST'))
def create_movies():
    if request.method == 'POST':
        title = request.form['title']
        genre = request.form['genre']

        if not title:
            flash('Title is required!')
        else:
            conn = get_db_connection()
            conn.execute('INSERT INTO movies (title, genres) values (?, ?)',
                        (title, genre))
            conn.commit()
            conn.close()
            return redirect(url_for('index'))

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

---

le code source est disponible sur : (lien github)

## Contact

lamia.kaciaissa@gmail.com    abiremahfouze@gmail.com

## Suivez-nous sur les réseaux sociaux



[www.simplon.co](http://www.simplon.co)

**SIMPLON**  
.CO