Flask - Les routes

Dans le développement web, l'une des premières choses que l'on doit maîtriser est la manière dont les requêtes sont gérées par le serveur, et comment elles sont redirigées vers le bon traitement. Lorsque l'on travaille avec Flask, un micro-framework pour Python, cet aspect est géré par un concept central : les **routes**. Cet article vous expliquera en détail ce que sont les **routes**, comment elles fonctionnent, et comment les utiliser efficacement dans vos applications web.

Qu'est-ce qu'une "Route" dans Flask ?

En termes simples, une **route** est une association entre une **URL** et une **fonction** Python dans votre application Flask. Lorsque Flask reçoit une requête HTTP, il regarde l'URL et essaie de faire correspondre cette URL à une route définie dans votre code. Si une correspondance est trouvée, Flask exécute la fonction associée et envoie la réponse générée au navigateur de l'utilisateur.

Exemple basique de route

Voici un exemple simple pour mieux comprendre ce concept :

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Bienvenue sur la page d'accueil !"

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

Dans cet exemple :

- `@app.route('/')` : C'est un décorateur qui lie l'URL racine (`/`) à la fonction `home()`.
- `home()` : Cette fonction est exécutée lorsque l'utilisateur visite l'URL racine du site web (par exemple, `http://localhost:5000/`). Elle renvoie une chaîne de caractères qui sera affichée dans le navigateur.

Chaque fois que Flask reçoit une requête pour l'URL `/`, il sait qu'il doit appeler la fonction `home()` et renvoyer la réponse correspondante.

Comment définir des routes en Flask ?

Utilisation des Décorateurs

Les routes en Flask sont principalement définies avec un **décorateur** : `@app.route()`. Ce décorateur est placé juste au-dessus d'une fonction, indiquant à Flask que cette fonction doit être exécutée lorsqu'un utilisateur accède à une URL spécifique.

Gérer plusieurs méthodes HTTP

Par défaut, une route en Flask ne répond qu'aux requêtes HTTP de type **GET**. Cela signifie que si un utilisateur entre une URL dans son navigateur ou clique sur un lien, Flask déclenchera la fonction liée à cette route avec une requête `GET`. Cependant, il est possible de spécifier d'autres méthodes HTTP, comme **POST**, **PUT**, ou **DELETE**, en fonction des besoins de votre application.

Voici un exemple où une route accepte à la fois les méthodes `GET` et `POST` :

© 2025 Mr. Lambrechts

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return "Tentative de connexion"
    return "Page de connexion"
```

Dans cet exemple :

- Si la méthode de la requête est **GET**, Flask renverra simplement "Page de connexion".
- Si la méthode est **POST**, par exemple lorsqu'un formulaire de connexion est soumis, la réponse sera "Tentative de connexion".

Les raccourcis `@app.get` et `@app.post`

Flask offre des **raccourcis** pour définir rapidement des routes qui utilisent spécifiquement les méthodes HTTP **GET** et **POST**. Plutôt que d'utiliser le décorateur général `@app.route()` et de préciser les méthodes acceptées via l'argument `methods`, vous pouvez utiliser les décorateurs `@app.get()` et `@app.post()`.

`@app.get()`

Ce raccourci est équivalent à `@app.route(..., methods=['GET'])`. Il est utilisé pour les routes qui acceptent uniquement les requêtes **GET**, typiquement pour afficher des pages ou des informations. Par exemple :

```python
@app.get('/')
def home():
    return "Page d'accueil"
```

Dans cet exemple, la route définie avec `@app.get('/')` accepte uniquement les requêtes GET, ce qui est idéal pour des pages statiques ou des ressources à lire.

`@app.post()`

De la même manière, `@app.post()` est un raccourci pour `@app.route(..., methods=['POST'])`. Ce décorateur est utilisé lorsque la route doit accepter uniquement des requêtes **POST**, souvent utilisées pour soumettre des formulaires ou envoyer des données au serveur. Voici un exemple :

```python
@app.post('/login')
def login():
    return "Traitement du formulaire de connexion"
```

Avec ce décorateur, la fonction `login()` ne sera accessible que pour les requêtes POST, ce qui est une bonne pratique pour les actions sensibles comme la soumission de formulaires.

Avantages des raccourcis

L'utilisation de ces raccourcis permet de rendre votre code plus **lisible** et plus **explicite** en ce qui concerne le type de requête HTTP que la route gère. Cela simplifie également la structure de votre code en éliminant la nécessité de spécifier l'argument `methods` pour les requêtes courantes GET et POST, ce qui est particulièrement utile dans des projets de petite à moyenne envergure.

Gestion des erreurs 404 avec des routes

En développement web, il est courant que les utilisateurs accèdent à des pages inexistantes, ce qui génère une erreur **404**. Flask vous permet de gérer cela facilement en définissant une route spéciale pour cette erreur :

```python
@app.errorhandler(404)
def page_not_found(error):
```

```
    return "Page non trouvée", 404
```

Ici, la fonction `page_not_found()` sera appelée chaque fois qu'une route non définie est accédée, et Flask renverra une réponse avec un message personnalisé et le code d'erreur HTTP 404.

Exemples complets d'application Flask avec des routes

Maintenant que nous avons vu plusieurs aspects des routes en Flask, voici un exemple d'application complète avec plusieurs routes :

```python
from flask import Flask, request

app = Flask(__name__)

# Route pour la page d'accueil
@app.route('/')
def home():
    return "Page d'accueil"

# Route dynamique pour afficher un profil utilisateur
@app.route('/user/admin')
def user_profile():
    return f"Profil de 'Admin'"

# Route pour gérer les connexions avec GET et POST
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return "Connexion réussie"
    else:
        return "Formulaire de connexion"

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

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

Ce que fait cette application :

- **Route racine (`/`)** : Renvoie "Page d'accueil".
- **Route utilisateur (`/user/admin`)** : Affiche le profil de l'utilisateur `admin`.
- **Route de connexion (`/login`)** : Affiche un formulaire de connexion pour une requête GET et traite la connexion pour une requête POST.
- **Gestion des erreurs 404** : Si un utilisateur tente d'accéder à une page non définie, il recevra un message "Cette page n'existe pas" avec un statut 404.

Conclusion

Les **routes** sont l'un des concepts les plus fondamentaux dans Flask et permettent de **définir comment votre application web doit répondre aux différentes requêtes HTTP**. En comprenant comment lier des URLs à des fonctions Python, gérer des paramètres dynamiques, et supporter différentes méthodes HTTP, vous serez capable de construire des applications web plus complexes et interactives.

Il est également important de faire attention à l'**ordre de définition des routes** dans votre application pour éviter des conflits entre des routes similaires, en définissant toujours les routes les plus spécifiques avant les routes plus générales.

Flask offre une grande flexibilité tout en restant léger et simple à utiliser. En maîtrisant l'utilisation des routes, vous poserez les bases nécessaires pour concevoir des applications web robustes et performantes. Bonne chance dans vos projets Flask !

© 2025 Mr. Lambrechts