Activité
Traitement des données d'un formulaire avec Flask

Dernière mise à jour le : 21/08/2023

Dans cette activité, on va illustrer comment un serveur Web peut récupérer et traiter les données d'un formulaire de connexion, présent sur de très nombreux sites.

Voici la vidéo associée à cette activité à partir de 7:33 :

Source : ▶ https://youtu.be/FdA1P7dY_18?start=453

Création de la route du formulaire de connexion

Dans le script `mon_app.py` on crée une route `/login` permettant d'accéder à notre formulaire via le template `login.html` :

mon_app.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/login")
def login():
    return render_template("login.html")

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

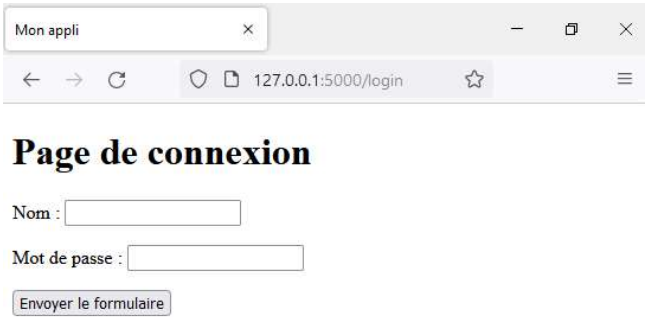
Le template `login.html`, situé dans le répertoire `templates` est le suivant :

templates/login.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mon appli</title>
</head>
<body>
  <h1>Page de connexion</h1>
  <form action="" method="">
    <p>
      <label for="nom">Nom : </label>
      <input type="text" name="nom" id="nom" required>
    </p>
    <p>
      <label for="mdp">Mot de passe : </label>
      <input type="password" name="mdp" id="mdp" required>
    </p>
    <p>
```

```
      <input type="submit" value="Envoyer le formulaire">
    </p>
  </form>
</body>
</html>
```

En naviguant à l'URL `http://127.0.0.1:5000/login` on obtient l'écran suivant :



Attributs `action` et `method`

Nouvelle route pour traiter les données

On choisit ici de traiter les données du formulaire avec une autre route : `/traitement`. Pour cela, on *ajoute* cette route dans notre script `mon_app.py` :

mon_app.py

```
@app.route("/traitement")
def traitement():
    return "Traitement des données"  # provisoire
```

La valeur de l'attribut `action` du formulaire doit être l'URL vers laquelle les données sont envoyées. Donc ici il faut que cela soit l'URL `/traitement`.

On pourrait donc écrire

templates/login.html

```
<form action="/traitement" method="">
```

mais on préfère utiliser la méthode `url_for` à qui on passe, sous forme d'une chaîne de caractères, le nom de la *fonction* associée à la route (dans notre cas c'est la fonction `traitement`) :

```
<form action="{{ url_for('traitement') }}" method="">
```

On écrit `url_for('traitement')` entre des doubles accolades pour que Flask évalue cette expression : ainsi `{{ url_for('traitement') }}` sera évaluée et remplacée par la route `/traitement` lors de la construction de la page (ce que vous pouvez voir en analysant le code source dans le navigateur).

Choix de la méthode

Ici, les données à transmettre sont à caractère confidentiel, donc on élimine tout de suite la transmission via l'URL (avec la méthode `GET`) et on choisit la méthode `POST` (même si on rappelle que c'est insuffisant pour sécuriser la transmission) :

templates/login.html

```
<form action="{{ url_for('traitement') }}" method="POST">
```

Traitement des données

Version minimale

On peut désormais écrire la version finale (ou presque) de la fonction `traitement()` :

```
# ne pas oublier d'importer les méthodes nécessaires
from flask import Flask, render_template, request

@app.route("/traitement", methods=["POST"])
def traitement():
    donnees = request.form
```

```
    nom = donnees.get('nom')
    mdp = donnees.get('mdp')
    if nom == 'admin' and mdp == '1234':
        return render_template("traitement.html", nom_utilisateur=nom)
    else:
        return render_template("traitement.html")
```

Analyse :

- Pour permettre à cette route d'accepter les requêtes de type `POST`, et donc de récupérer les données du formulaire, on a définit `["POST"]` comme valeur du paramètre `methods` du décorateur `@app.route("/traitement")`.
- Pour récupérer les données d'un formulaire avec Flask, on utilise `request.form` qui est un dictionnaire dont :
  - les clés sont les noms des paramètres transmis (qui sont les valeurs des attributs `name` des champs du formulaire)
  - les valeurs sont les valeurs de ces paramètres
- On peut donc récupérer le nom et le mot de passe saisis grâce à `donnees.get('nom')` et `donnees.get('mdp')` puis effectuer le traitement souhaité.
- Ici, le traitement se limite à n'autoriser qu'une seule personne à se connecter, mais en théorie il faudrait chercher dans un fichier ou dans une base de données si le nom et le mot de passe correspondent à un utilisateur connu.
- Dans le cas où l'utilisateur est connu on transmet au template son nom pour pouvoir l'afficher à l'écran, dans le cas contraire non.

Dans le dossier `templates`, on crée le template `traitement.html` contenant :

templates/traitement.html

```
<body>
  {% if nom_utilisateur %}
    <p>Bonjour {{ nom_utilisateur }}, vous êtes connecté.</p>
  {% else %}
    <p>Un problème est survenu.</p>
  {% endif %}
</body>
```

Analyse : Si la fonction `traitement()` a transmis `nom_utilisateur` au template (on sait que l'identification est OK et on passe dans le `if`), alors on l'utilise pour afficher un message de connexion réussie, sinon (si l'identification a échoué, et on passe dans le `else`) on affiche un paragraphe d'erreur.

Améliorations

> Ces améliorations ne sont pas nécessaires pour comprendre le mécanisme de traitement des données d'un formulaire côté serveur.
> On les présente néanmoins ici car elles le sont dans la vidéo et que ce sont des choses intéressantes à savoir lorsque l'on veut commencer à développer de petites applications web (avec Flask ou tout autre framework de développement web).

Gérer l'accès direct à la route de traitement

On peut améliorer la fonction `traitement()` de sorte à ce qu'un utilisateur naviguant à l'URL `http://127.0.0.1:5000/traitement` sans passer par le formulaire, donc avec une méthode `GET`, soit redirigé vers la page d'accueil du site. En effet, dans ce cas aucune donnée n'est transmise, il n'est donc pas nécessaire ni logique d'effectuer les traitements précédents.

Pour cela, on ajoute `"GET"` dans les méthodes autorisées pour la route `/traitement` et on utilise `request.method` pour déterminer la méthode utilisée pour la requête :

- si elle vaut `"POST"`, on fait le traitement précédent
- sinon (c'est qu'il s'agit d'une requête `GET`) on redirige l'utilisateur grâce à la méthode `redirect` à qui on passe l'URL correspond à la fonction `index` grâce à `url_for` :

mon_app.py

```
# ne pas oublier d'importer les méthodes nécessaires
from flask import Flask, render_template, request, redirect, url_for

@app.route("/traitement", methods=["POST", "GET"])
def traitement():
    if request.method == "POST":
        donnees = request.form
        nom = donnees.get('nom')
        mdp = donnees.get('mdp')
        if nom == 'admin' and mdp == '1234':
            return render_template("traitement.html", nom_utilisateur=nom)
        else:
            return render_template("traitement.html")
```

```
    else:
        return redirect(url_for('index'))
```

Ajouter un lien vers la page de connexion

Même si ce n'est pas utile pour le propos de cette activité, on peut en profiter pour ajouter, sur la page d'accueil, un lien vers la page de connexion. C'est très simple, on ajoute une balise de lien hypertexte `<a>` dont la valeur de l'attribut `href` est l'URL vers laquelle pointe l'hyperlien. En utilisant `url_for` on obtient facilement l'URL (ici `/login`) associée à la fonction `login` :

templates/login.html

```
<body>
  <p>Bienvenue sur mon site !</p>
  <a href="{{ url_for('login') }}">Se connecter</a>
</body>
```

On aurait aussi pu écrire directement le lien avec l'URL en dur :

```
<a href="/login">Se connecter</a>
```

À vous de jouer ! ✍

Exercice 1

Créer un formulaire permettant à un utilisateur de saisir son nom, son prénom et de choisir dans une liste d'options la spécialité qui sera abandonnée parmi les spécialités "HGGSP", "HLP", "NSI", "SVT", "Mathématiques", "Physique-Chimie", "SES", "LLCE".

Les données saisies devront être transmises au serveur via la méthode POST et le serveur devra construire un template avec un message personnalisé en guise de réponse. Par exemple, si l'utilisateur saisit le prénom "Armin", le nom "Ronacher" et sélectionne la spécialité "SES", alors la page renvoyée doit contenir un paragraphe indiquant : "Bonjour Armin Ronacher, vous avez choisi de ne pas poursuivre la spécialité SES en Terminale".

Exercice 2 (optionnel)

L'idée de cet exercice est de simuler le fonctionnement d'un dictionnaire en ligne. On supposera qu'il s'agit d'un dictionnaire Français-Anglais.

Voici les contraintes à respecter :

- L'utilisateur accède à une page contenant un formulaire permettant de saisir un mot en français.
- Les données du formulaire doivent être passées au serveur via la méthode POST.
- Le serveur récupère le mot saisi et recherche la traduction dans le (petit) dictionnaire `traductions` suivant

```
traductions {
"mot": "word",
"supprimer": "remove",
"requête": "request",
"méthode": "method",
"répertoire": "directory",
}
```

- Le serveur construit une page de réponse (un template) contenant :
  - le mot et sa traduction si le mot est trouvé
  - un message indiquant que le mot n'est pas trouvé sinon.

Références :

- Documentation officielle de Flask : https://flask.palletsprojects.com/en/2.0.x/

Germain BECKER & Sébastien POINT, Lycée Emmanuel Mounier, ANGERS

Voir en ligne : info-mounier.fr/premiere_nsi/web/activite-traitement-formulaire