Client Serveur avec Flask - TD2
04 Avril 2024

1 Introduction

Le but de ce TD va être de créer un serveur avec la bibliothèque Flask de Python et d'interagir avec notre page web.

Téléchargez l'archive ZIP nommé "Exercice-TD2-IHM" disponible au lien suivant : https ://github.com/LoicBlanchard7/IHM-NSI-TD2

Vous trouverez dans cette archive plusieurs fichiers, commencez en ouvrant le fichier "views.py" dans votre IDE. (Genius, Spyder, Visual Studio Code, ...)

2 Création du serveur

Pour créer notre serveur, nous allons suivre quelques étapes :

Dans votre fichier "views.py", nous allons commencer par importer la bibliothèque Flask qui va nous permettre de créer notre serveur.

```
1 import flask
```

Ensuite, nous allons créer un objet app. Cet objet est nécessaire car il va constituer notre application.

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

A la suite de votre programme, nous allons initialiser un décorateur. Ce décorateur permettra d'initialiser la racine de notre site.

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

Nous allons ensuite créer notre première page web grâce à Flask :

```
1 def index():
2     return "<p>Notre premier site avec Flask !</p>"
```

Pour terminer notre programme nous allons ajouter la ligne ci-dessous, permettant de lancer le serveur.

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

Après avoir exécuté le programme ci-dessus, ouvrez votre navigateur web et tapez dans la barre d'adresse "localhost :5000".

Vous devriez avoir la phrase "Notre premier site avec Flask !" qui s'affiche dans votre navigateur.

Notre serveur et notre client se trouvent sur la même machine, avec le "localhost", on indique au navigateur que le serveur web se trouve sur le même ordinateur que lui (on parle de machine locale). Dans un cas normal, la barre d'adresse devrait être renseignée avec l'adresse du serveur web. Le "5000" indique le port, nous n'étudierons pas cet aspect des choses ici.

3 Une autre page

A la suite de votre programme, nous allons initialiser un second décorateur. Ce décorateur permettra de naviguer sur une seconde page de notre site.

```
1 @app.route('/explications')
2 def explications():
3   return "<p>Voici notre page d'explications !</p>"
```

Attention : la dernière ligne de votre programme devra impérativement être "app.run(debug=True)" !

Après avoir exécuté le programme ci-dessus, saisissez "localhost :5000/explications" dans la barre d'adresse de votre navigateur.

Comme vous pouvez le constater, le serveur nous renvoie dans ce cas une autre page. Évidemment l'URL racine ("/") reste disponible, vous pouvez passer d'une page à l'autre en modifiant l'URL dans la barre d'adresse ("localhost :5000" ou "localhost :5000/explications").

4 Les templates

Comme nous l'avons vu, écrirez le code HTML dans le programme Python n'est pas pratique. C'est pourquoi il existe une alternative avec Flask.

Dans l'archive que vous avez téléchargé au début du TD se trouve un dossier "templates". Dans ce répertoire, créez un nouveau fichier "index.html".

Dans votre body ajouter un titre (h1) ainsi qu'un paragraphe (p).

Retournois maintenant dans notre fichier "views.py". Maintenant que nous avons créer un "template", nous allons l'utiliser.
Modifions la ligne :

```
1 return "<p>Notre premier site avec Flask !</p>"
```

par

```
1 return render_template("index.html")
```

Relancez le programme Python et tapez "localhost :5000" dans la barre d'adresse de votre navigateur. Le serveur renvoie maintenant au client la page HTML correspondant au fichier "index.html" qui a été créé dans le répertoire "templates". Attention, les fichiers HTML devront systématiquement se trouver dans un répertoire nommé "templates".

5 D'un site statique à un site dynamique

Pour l'instant notre site est statique : la page reste identique, quelles que soient les actions des visiteurs. Flask permet de créer des pages dynamiques. Pour celà, nous allons apporter des modifications à nos fichiers. Le but va être d'afficher l'heure à laquelle vous avez ouvert la page principale du site.

Dans votre fichier "views.py", ajoutez tout en haut de votre programme la ligne ci-dessous.

```
1 import datetime
```

Puis, modifiez l'initialisation de la page index pour que celà corresponde avec le code ci-dessous :

```
1 @app.route('/')
2 def index():
3   date = datetime.datetime.now()
4   h = date.hour
5   m = date.minute
6   s = date.second
7   return render_template("index.html", heure = h, minute = m,
8 seconde = s)
```

Dans le programme ci-dessus nous importons le module "datetime" afin de pouvoir déterminer la date et l'heure courante. Puis grâce nous récupérons les valeurs de l'heure, des minutes et des secondes dans trois variables h, m et s. La dernière ligne "render_template" contient 3 paramètres que nous allons pouvoir utiliser dans le fichier HTML.

Pour afficher l'heure sur notre page web, nous allons devoir modifier notre page "index.html" se trouvant dans le dossier "templates".

Pour se faire, nous allons ajouter la ligne suivante dans le body :

```
1 <p>L'heure actuelle est {{heure}} h {{minute}} minutes et
2 {{seconde}} secondes</p>
```

Testez ces modifications en saisissant "localhost :5000" dans la barre de votre navigateur web.

Comme nous pouvons le voir, l'heure de change pas automatiquement. En effet, nous avons bien une page dynamique puisqu'à chaque fois que vous actualisez la page dans votre navigateur, l'heure courante s'affiche. Néanmoins, l'heure s'actualise à chaque fois que vous actualisez la page car vous effectuez une nouvelle requête et en réponse à cette requête, le serveur envoie une nouvelle page HTML.

Durant le cours précédent nous avions vu qu'il n'était pas possible de créer du dynamique seulement avec de l'HTML. Le fichier "index.html" ne contient donc pas du HTML car les paramètres en {{}} n'existent pas en HTML. Le fichier contient un langage de template nommé Jinja ressemblant à du HTML mais apportant de nombreuses fonctionnalités.

6 Formulaire

Dans notre dossier "templates", vous pourrez trouver deux autres fichiers : "formulaire.html" et "resultat.html".

Nous allons donc commencer par créer un nouveau décorateur à notre fichier "views.py". Pour se faire, reprenez l'exemple du fichier index pour créer un nouveau décorateur nous

permettant d'accèder à la page "formulaire.html".

Analysons maintenant ce qui se trouve dans le fichier "formulaire.html". (Ouvrez ce fichier dans votre IDE).

6.1 Analyse

Nous effectuons une requête HTTP avec l'URL "/formulaire", le serveur génère une page web à partir du fichier "formulaire.html", cette page, qui contient un formulaire est alors envoyée vers le client.

On remarque 2 attributs dans cette balise form : action="http ://localhost :5000/resultat" et method="post". Ces 2 attributs indiquent que le client devra effectuer une requête de type POSTdès que l'utilisateur appuiera sur le bouton "Envoyer". Cette requête POST sera envoyée à l'URL "http ://localhost :5000/resultat". Les données saisies dans le formulaire seront envoyées au serveur par l'intermédiaire de cette requête.

6.2 Réception méthode POST

Maintenant que nous savons comment les données du fichier vont être envoyées, nous allons nous intéresser à la réception de ces données.

Pour cela, nous allons ajouter ce bout de programme à la suite de notre fichier "views.py".

```
app.route('/resultat',methods = ['POST'])
def resultat():
  result = request.form
  n = result['user_name']
  p = result['user_mail']
  m = result['user_message']
  return render_template("resultat.html", nom=n, mail=p,
message=m)
```

"request.form" est un dictionnaire Python qui a pour clés les attributs "name" des balises "input" du formulaire et comme valeurs ce qui a été saisi par l'utilisateur. Si l'utilisateur saisit "Blanchard", "loicblanchard@gmail.com" et "Nous sommes en cours !", le dictionnaire "request.form" sera :

```
{'user_name':'Blanchard', 'user_mail':'loicblanchard@gmail.com',
'user_message':'Nous sommes en cours !'}
```

Nous allons maintenant essayer d'afficher ces informations dans la page "resultat.html". Commencez en ouvrant le fichier, puis modifier le pour afficher les résultats. Pour vous aider, vous pouvez reprendre ce que l'on a vu dans le fichier "index.html".

6.3 Réception méthode GET

Pour gérer le formulaire, il est possible d'utiliser une méthode HTTP "GET" à la place de la méthode "POST" :

Pour utiliser la méthode "GET", ouvrez le fichier "formulaire.html" et modifier la méthode d'envoie du formulaire en passant de "post" à "get".

Dans le fichier "views.py", changez la méthode également de "GET" à "POST" ainsi que la ligne de code suivante :

```
result=request.form
```

par

```
result=request.args
```

Relancez l'exécution de "views.py" et saisissez "localhost :5000/formulaire" dans la barre d'adresse d'un navigateur web. Remplissez le formulaire et envoyez le. Vous avez dû remarquer que cette fois-ci, les informations du formulaire sont transmises au serveur par l'intermédiaire de l'URL :

localhost :5000/resultat ?user_name=Blanchard&user_mail=loicblanchard@gmail.com& user_message=coucou

Dans le cas de l'utilisation d'une méthode "POST" les données issues d'un formulaire sont envoyées au serveur sans être directement visibles, alors que dans le cas de l'utilisation d'une méthode "GET", les données sont visibles (et accessibles) puisqu'elles sont envoyées par l'intermédiaire de l'URL.

7 Pour aller plus loin

Ajoutez la ligne de code ci-dessous en haut du body de chacun de vos fichiers HTML :

```
{% include 'navbar.html' %}
```

Vous remarquerez qu'il est possible d'injecter du code HTML dans les pages HTML. Dans notre cas, nous avons injecté une barre de navigation permettant à l'utilisateur de naviguer entre les pages web de son site !