université
PARIS-SACLAY

Programmation en Python
Master 2 Réseaux Télécoms
TP 11

1 Nom complet sur la page personnelle

Si c'est pas encore fait, ajouter des noms et prénoms aux utilisateurs fictifs de notre application. Modifiez le code du gabarit `home.html` de sorte qu'il affiche en tête de la page le nom et le prénom de l'utilisateur au lieu de son login, par exemple :

Page personnelle de Jean Dupont

N.B. : le nom et le prénom correspondent aux attributs `last_name` et `first_name` du modèle `User`.

2 Un formulaire pour ajouter un outil

Les formulaires HTML (les éléments du code délimités pas les balises `<form>` et `</form>`) permettent à l'utilisateur d'envoyer les donnés vers une application Web dynamique. L'API de Django contient le module `forms` qui facilite le travail avec des formulaires. Notamment, la classe `Form` de ce module décrit un formulaire avec des champs d'entrée typés, ce qui permet de générer automatiquement le code de validation des valeurs soumises.

2.1 Une classe pour un formulaire

Nous allons utiliser une classe dérivée de `Form` pour permettre à l'utilisateur d'ajouter un outil dans la base de données :

```
from django import forms

class AddToolForm(forms.Form):
```

```
description = forms.CharField(max_length=256,
                              label="Description")
price=forms.DecimalField(decimal_places=2,
                         max_digits=6, label="Prix")
```

Notez que les contraintes sur les champs de ce formulaire sont conformes à celles du modèle `Tool`. Créez un nouveau fichier `forms.py` dans le répertoire `coop` et ajoutez le code ci-dessus dans ce fichier.

2.2 Injecter un formulaire dans le contexte d'un gabarit

Pour afficher un formulaire sur une page Web, le gabarit HTML doit avoir l'accès au code de ce premier. Modifiez le code de la fonction `home` dans le fichier `views.py` de la façon suivante :

```
def home(request):
    addform = AddToolForm()
    context = {'addform': addform}
    return render(request, "coop/home.html", context)
```

Notez que la fonction `render` prend maintenant un troisième argument positionnel nommé `context`. Cet argument contient l'information supplémentaire nécessaire pour générer une page Web. Pour importer la classe `AddToolForm`, ajoutez l'instruction suivante aux commandes d'importation :

```
from .forms import AddToolForm
```

(notez que le point devant le nom du module signifie que Python va chercher ce dernier dans le même répertoire que le fichier `views.py`).

2.3 Affichage du formulaire

Notre formulaire est maintenant accessible dans le gabarit `home.html` sous forme d'un token `addform`. Ajoutez le code suivant dans le fichier `home.html`, juste après la table des outils :

```
<h2>Ajouter un outil:</h2>
<form action="/coop/add-tool/" method="post">
  {% csrf_token %}
  <table>
    {{ addform.as_table }}
  </table>
  <input type="submit" value="Ajouter"/>
</form>
```

On remarque dans ce code la ligne

```
{% csrf_token %}
```

qui sert à prévenir les attaques de type *cross-site request forgery*. On note également que les balises correspondantes aux champs d'entrée sont générés automatiquement grâce à la ligne suivante :

```
{{ addform.as_table }}
```

Naviguez vers `http://localhost:8000/coop/` pour visualiser le nouveau formulaire. Vérifiez que le champ du prix accepte uniquement les valeurs décimales avec au plus deux chiffres après l virgule et que le champ de la description ne peut pas être vide. Par contre, on ne peut pas encore ajouter un outil, même si tous le champs sont correctement remplis, car on n'a pas défini l'action correspondant à l'URL du formulaire.

3 Connecter un formulaire avec une view

Ajoutez dans le fichier `views.py` la fonction suivante :

```
def add_tool(request):
    form = AddToolForm(request.POST)
    if form.is_valid():
        pass
    return redirect('home')
```

Il faut maintenant lier l'URL du formulaire (tel qu'il est spécifié dans l'attribut `action` de ce dernier) avec la fonction `add_tool`. À cette fin, on ajoute l'élément suivant dans la liste `urlpatterns` dans le fichier `coop/urls.py` :

```
path('add-tool/', views.add_tool, name = "add-tool")
```

Maintenant, quand on soumet le formulaire, Django appelle la fonction `add_tool`. Vérifier ce fait en remplaçant la commande `pass` dans le corps de cette fonction par une instruction `print` (le texte imprimé doit apparaitre dans le terminal du serveur).

4 Ajout d'un outil

L'appel de la méthode `is_valid` de la classe `AddToolForm` sert à valider le formulaire et récupérer ses données. Les données validées sont stockées dans l'attribut `cleaned_data` de la classe, sous forme d'un dictionnaire indexé par les noms des champs. Affichez ce dictionnaire dans le terminal du serveur, en ajoutant la commande correspondante dans la fonction `add_tool`.

FIGURE 1 – Page principale de l'application.

On peut maintenant créer un objet représentant le nouvel outil, en passant les paramètres `description` et `prix` comme les arguments (nommés) dans le constructeur de la classe `Tool`. Pour spécifier le paramètre `owner`, on utilisera l'identité de l'utilisateur connecté, accessible comme l'attribut `user` de l'objet `request`. Pour sauvegarder l'outil dans la base de de données on utilisera la méthode `save`, héritée de la classe `Model`. Ajoutez le code correspondant dans la fonction `add_tool` et vérifiez que l'outil ajouté apparait réellement dans la table.

5 Travailler avec des formulaires brutes

Les formulaires gérés par Django ne sont pas adaptés au cas où les champs d'entrée sont dynamiques, comme les cases à cocher sur la Figure 1. Dans ce cas, l'information soumise par l'utilisateur peut être retrouvée directement dans l'attribut `POST` de l'argument `request` de la view correspondante.

5.1 Un formulaire pour supprimer des outils

Modifiez le code du gabarit `home.html` en entourant la table des outils par les balises d'un nouveau formulaire :

```
<form action="/coop/delete-tools/" method="post">
  {% csrf_token %}
  ...
```

```
</form>
```

Ajoutez à la table la quatrième colonne avec des cases à cocher et le bouton de commande placé dans l'entête (voir Figure 1). Pour une case à cocher, on utilisera la balise suivante :

```
<input type="checkbox" value="{{tool.id}}" name="to_delete"/>
```

Vérifiez que la table modifiée s'affiche correctement.

5.2 Suppression des outils

Ajoutez la fonction suivante dans le fichier `views.py` :

```
def delete_tools(request):
    # Code a ajouter
    return redirect('home')
```

Reliez cette fonction avec le formulaire en ajoutant un élément dans la liste `urlpatterns` dans le fichier `coop/urls.py`.

La liste des identifiants des outils à supprimer est accessible dans la fonction `delete_tools` avec l'expression suivante :

```
request.POST.getlist('to_delete')
```

Notez que l'argument `'to_delete'` correspond au nom de la case à cocher dans la balise `<input>` correspondante. Cette expression retourne une liste de chaînes de caractères, qu'il vous convient à convertir en entiers et utiliser pour supprimer les outils. On utilisera la méthode `filter` de l'objet `Tool.objects` pour sélectionner les outils concernés, suivie de la méthode `delete`.

6 Résolution inverse des URL

Pour être portable, une application Django ne doit pas contenir les mentions de son propre nom dans son code (notez que le répertoire `templates/coop` ne contredit pas cette règle car dans ce cas il s'agit d'un espace de noms et pas du nom de l'application). De ce point de vue la ligne

```
<form action="/coop/add-tool/" method="post">
```

pose un problème. Pour y remédier, on peut utiliser la résolution inverse des URL, avec la balise Django suivante :

```
<form action="{% url 'add-tool' %}" method="post">
```

Notez que le nom `'add-tool'` correspond à l'argument nommé `name` dans l'appel de la fonction `path` dans le fichier `coop/urls.py`.

Remplacez les actions des deux formulaires de notre application avec les balises de résolution inverse et vérifiez que l'application fonctionne toujours.