TP 4 : Technologies Web II [côté serveur]

Formulaires

Objectifs du TP

Le but principal de ce TP est de découvrir comment mettre en place les formulaires dans Django.

Consigne

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile!
N'hésitez pas à chercher des informations sur les fonctions sur Internet, un bon informaticien doit savoir chercher des informations en ligne. Vous retrouverez notamment la documentation Django ici : `https://docs.djangoproject.com/en/4.1/`

Attention aux "copier-coller" du PDF vers votre code, des différences d'espace et d'indentation peuvent avoir lieu.
Il est donc préférable d'écrire vous même les commandes et le code demandés (c'est relativement court).

Introduction

Dans les TP précédents, nous avons vu l'ensemble des éléments du modèle MVC dans Django. Cela nous a notamment permis de mettre en place un modèle de données avec les modèles, de gérer ces données ainsi que le comportement du site avec les contrôleurs et de restituer des pages avec les données grâce aux vues.

Cependant, un élément important de la programmation Web nous manque : l'interaction de l'utilisateur avec le site Web. Cette interaction peut correspondre à la validation d'un utilisateur, à la réponse à des questions, etc. Ces actions entraînent le plus souvent une modification des données.

En programmation Web, cette interaction se caractérise principalement par les formulaires. Nous allons donc voir comment mettre en place des formulaires dans Django.

1 Gestion des formulaires HTML en Django

1.1 Formulaire minimal

Prenons l'exemple de notre application de sondage avec un formulaire HTML pour choisir une réponse parmi les choix possibles et valider le choix. Le formulaire HTML pourrait par exemple ressembler à ça :

```
<form method="post">
<fieldset>
    <legend><h1>Participerez-vous a la nuit de l'Info cette annee 2022 ?</h1></legend>
        <input type="radio" name="choice">
        <label>Oui evidemment</label><br>
        <input type="radio" name="choice">
        <label>Je ne sais pas encore</label><br>
</fieldset>
<input type="submit" value="Vote">
</form>
```

Evidemment, nous n'allons pas pouvoir coder ce type de formulaire "en dur" pour chaque question. L'idée est alors d'utiliser les templates dans Django. Nous allons donc modifier le fichier **polls/-details.html** pour y ajouter le formulaire HTML. Modifiez le template **polls/details.html** pour y ajouter le formulaire HTML.

1- La première chose à faire est d'ajouter le titre de la question de manière dynamique grâce au langage de template dans Django :

```
    <legend><h1>{{ question.question_text }}</h1></legend>
```

2- Ensuite pour les choix, utilisez également le langage de template :

```
    {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
        <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
    {% endfor %}
```

Cela parcourt les différents choix pour la question. Pour chaque choix, un bouton radio est créé avec un id correspondant à son indice dans la boucle (**forloop.counter**). L'attribut value de chaque bouton radio correspond à l'ID du vote choisi. Le nom (name) de chaque bouton radio est "choice". Cela signifie que lorsque quelqu'un sélectionne l'un des boutons radio et valide le formulaire, les données `POST choice=#` (où # est l'identifiant du choix sélectionné) seront envoyées.

3- Pour éventuellement gérer le cas où aucun choix n'est sélectionné lors de la validation, ajoutez également une ligne qui permet d'afficher un message d'erreur dans ce cas. La gestion de cette erreur sera vue plus tard. Le code pour le `<fieldset>` ressemble maintenant à ceci :

```
<fieldset>
    <legend><h1>{{ question.question_text }}</h1></legend>
    {% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
    {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
        <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
    {% endfor %}
</fieldset>
```

4- Définissez également l'URL qui traitera l'envoi du formulaire avec l'attribut `action` de la balise `form`. Dans notre cas, ce sera l'url correspondant à la vue **vote** (l'url et la vue ont déjà été créées au TP précédent) :

```
<form action="{% url 'polls:vote' question.id %}" method="post">
```

5- Comme le formulaire utilise la méthode `POST` permettant potentiellement de modifier des données, il est nécessaire de prévenir les attaques inter-sites. Django facilite également cela grâce à la balise de gabarit `{% csrf_token%}` à ajouter à tous les formulaires `POST` :

```
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
```

6- Après les différentes étapes précédentes, le formulaire intégré au template doit ressembler à ça :

```
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
<fieldset>
    <legend><h1>{{ question.question_text }}</h1></legend>
    {% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
    {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
        <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
    {% endfor %}
</fieldset>
<input type="submit" value="Vote">
</form>
```

1.2 Récupération et traitement des données

L'intérêt des formulaires est de pouvoir récupérer les informations rentrées par les utilisateurs. Cette récupération et cette gestion des données provenant des formulaires fait partie du comportement du site Web, c'est dans les vues (contrôleurs) que cela se passe.

Pour notre exemple, il s'agit de la vue **vote()** créée au TP 3, de manière très simple. Essayons de modifier cette vue pour qu'elle prenne en considération les votes des utilisateurs. Reprenez la vue **vue()** dans le fichier **views.py** de l'application de sondage.

1- La première chose à faire est de récupérer la question dont on souhaite modifier le nombre de votes

```
from django.http import HttpResponse
from django.shortcuts import get_object_or_404, render

from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
```

2- Ensuite, il faut récupérer le choix de l'utilisateur. Cela se fait via **request.POST** qui renvoie un dictionnaire contenant les données du formulaires. Dans notre cas, le nom utilisé pour envoyer la réponse dans le formulaire HTML est `choice`. Nous utiliserons alors la clé `choice` dans le dictionnaire :

```
request.POST['choice']
```

Il est possible que l'utilisateur envoie le formulaire sans avoir sélectionné de choix (nous avions prévu cela dans le template). Pour gérer ce cas là, nous allons passer par un **try except** en Python. On essaye une opération avec le **try** et on lève une exception dans le cas où cela échoue avec le **except**. Modifiez la suite de la vue comme ceci :

```
from django.http import HttpResponse
from django.shortcuts import get_object_or_404, render

from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
     })
```

Dans le cas où l'utilisateur n'as pas fait de choix, la page de formulaire est ré-envoyée avec un variable correspondant au message d'erreur pour être interprétée par le template.

3- Dans le cas où aucune exception n'est levée, il faut alors mettre à jour la base de données en comptabilisant le vote. Une fois le vote comptabilisé et la base de données mise à jour, il faut renvoyer une réponse. En programmation Web, une bonne pratique consiste à ne pas envoyer du contenu HTML directement après une requête POST mais de rediriger vers une page existante (cela permet notamment d'éviter de traiter deux fois la requête). Pour cela, Django propose les **HttpResponseRedirect**, qui prend en paramètre l'URL de la page de redirection (Il sera donc nécessaire d'également importer **HttpResponseRedirect**. De plus, une bonne pratique Django consiste à ne pas directement coder en dur l'URL de la page mais de passer par le comportement d'une vue. Il est préférable plutôt de donner une vue que l'on souhaite utiliser pour la redirection, ainsi que la partie variable de l'URL. Cela se fait par la méthode **reverse()**, qu'il faudra importer. Modifiez la suite de la vue comme ceci :

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse
from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
     })
     else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

4- La redirection fait appel à la vue **results**, créez-la comme ceci :

```
def results(request, question_id):
  question = get_object_or_404(Question, pk=question_id)
  return render(request, 'polls/results.html', {'question': question})
```

5- Pour rendre le résultats plus intéressant, en prenant notamment en compte la mise à jour du nombre de vote, créez le template **polls/results.html** ci dessous et appelez le dans la vue **results()** :

```
<h1>{{ question.question_text }}</h1>
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.nb_votes }} vote(s)</li>
{% endfor %}
</ul>
<a href="{% url 'polls:detail' question.id %}">Vote again?</a>
```

2 Les formulaires de Django

Dans la section précédente, nous avons vu comment utiliser les formulaires HTML avec Django. Comme beaucoup d'autres choses que nous avons vu, Django permet également de faciliter la création de formulaires avec une classe prédéfinie **Form**. Comme pour les modèles, la classe **Form** décrit la structure d'un formulaire, son comportement et son apparence.

Nous allons voir l'utilisation de la classe **Form** pour la création d'un formulaire de contact. Pour faciliter la gestion et éventuellement la réutilisation, nous allons créer une nouvelle application Web dédiée à ce formulaire.

1- Créer une nouvelle application appelée **myform** et ajoutez-la dans les applications installées du fichier **settings.py**

2.1 Un premier formulaire de contact

1- Pour créer notre premier formulaire de contact, nous allons créer un nouveau fichier Python dédié (il serait possible d'écrire le formulaire dans **views.py** mais cela mélangerait plusieurs informations). Créez le fichier **myforms.py** au même endroit que **views.py** puis écrivez le code suivant :

```
from django import forms

class ContactForm(forms.Form):
  name = forms.CharField(max_length=200)
  firstname = forms.CharField(max_length=200)
  email = forms.EmailField(max_length=200)
  message = forms.CharField(max_length=1000)
```

Cela crée un classe de formulaire héritant de **Form** avec des champs.

2- La génération du formulaire correspond à un comportement du site Web. Cela se fera donc dans le fichiers **views.py**. Définissez une fonction simple permettant d'afficher le formulaire (il faut importer le fichier où est décrit le formulaire) :

```
from django.shortcuts import render
from .myforms import ContactForm

def contact(request):
  contact_form = ContactForm()
  return render(request, 'myform/contact.html', {'contact_form' : contact_form})
```

3- Il faut maintenant créer les routes (URL) correspondantes pour l'affichage du formulaire. Tout d'abord, dans le fichier **urls.py** de `ensisa_project`, ajoutez la ligne suivante :

```
path('contacts/',include('myform.urls')),
```

Puis, créez un nouveau fichier **urls.py** dans l'application `myform` avec le code suivant :

```
from django.urls import path
from . import  views

app_name = 'myform'
urlpatterns=[
  path('',views.contact,name='contact'),
]
```

4- Il ne reste maintenant plus qu'à créer le template **contact.html** dans `myform/templates/myform/` avec le code suivant :

```
{{ contact_form.as_p }}
```

La méthode **as_p** appelée sur le formulaire signifie simplement que le formulaire sera affiché avec des balises <p>

5- Lancez le serveur et rendez-vous à l'URL : `http://127.0.0.1:8000/contacts/`, le formulaire a été généré automatiquement!

2.2 Un deuxième formulaire à partir d'un modèle

Un des avantages des formulaires dans Django est qu'il est possible de dériver un formulaire à partir d'une classe d'un modèle (un **ModelForm**) . Par exemple, imaginez que pour votre application, vous ayez créé une class `User` décrivant un utilisateur. Il serait intéressant à un moment d'avoir un formulaire permettant par exemple à un nouvel utilisateur de s'inscrire en renseignant ses informations. Plutôt que re-coder le formulaire avec les mêmes champs que le modèle, il est plus simple de le dériver.

1- Créez le modèle de `User` dans **models.py** :

```
from django.db import models

class User(models.Model):
  name = models.CharField(max_length=200)
  firstname = models.CharField(max_length=200)
  email = models.EmailField(max_length=200)
  message = models.CharField(max_length=1000)
```

2- Effectuez les migrations pour prendre en compte la création du modèle

3- Dans le fichier **myforms.py**, créez une seconde classe de formulaire à partir d'un modèle (un **ModelForm**) (il faut importer le modèle) comme ceci :

```
from django.forms import ModelForm
from .models import User

class ContactFormModel(ModelForm):
  class Meta:
    model = User
    fields = ('name', 'firstname', 'email', 'message')
```

4- Modifiez la vue pour ajouter le second formulaire :

```
from django.shortcuts import render
from .myforms import ContactForm, ContactFormModel

def contact(request):
  contact_form = ContactForm()
  contact_form_from_model = ContactFormModel()
  return render(request, 'myform/contact.html', {'contact_form' : contact_form,
  'contact_form_from_model' : contact_form_from_model})
```

5- Modifiez également le template pour avoir les deux formulaires :

```
{{ contact_form.as_p }}
<br/>
{{ contact_form_from_model.as_p }}
```

6- Re-lancez le serveur et rendez-vous à l'URL `http://127.0.0.1:8000/contacts/` pour voir l'affichage des deux formulaires identiques mais générées de deux manières différentes.

N'hésitez pas à creuser plus loin les formulaires **Form** ici : `https://docs.djangoproject.com/fr/4.1/topics/forms/`, ou les formulaires **ModelForm** ici : `https://docs.djangoproject.com/fr/4.1/topics/forms/modelforms/`.