Modules

CrispyForms

Modules views.py

Nous utilisons différents modules:

```
from django.shortcuts import render, redirect
from django.views import View
from django.http import HttpResponseRedirect
from django.urls import reverse_lazy
from .models import Post, Comment, UserProfile
from .forms import PostForm, CommentForm
from django.views.generic.edit import UpdateView, DeleteView
from django.contrib.auth.mixins import UserPassesTestMixin, LoginRequiredMixin
```

Render

```
return render(request, 'social/post_detail.html', context)
```

Render est un module de django.shortcuts

Les moteurs de rendu personnalisés doivent implémenter une méthode render(template_name, context, request=None). Cette méthode doit renvoyer un modèle rendu (sous la forme d'une chaîne de caractères) ou soulever la question `TemplateDoesNotExist`.

View

Le classe générique `View` est une classe parent pour les vues, qui possède différentes méthodes utiles pour celles-ci.

https://docs.djangoproject.com/en/4.1/ref/class-based-views/base/

Reverse_lazy()

```
def get_success_url(self):
        pk = self.kwargs['pk']
        return reverse_lazy('post-detail', kwargs={'pk': pk})
```

Fonction utile pour procéder à une résolution d'URL `avant le chargement` de la configuration des `URLs`.

Dans notre projet, nous l'utilisons pour les mises à jour et supression de posts et commentaires, ou encore dans la création d'un profil

Kwargs et Args

```
 def post(self, request, pk, `args, ``kwargs):
        ...
```

`args` permet de passer plusieurs arguments à une fonction. Si on ajoute print(args), cela va retourner un tuple contenant tous les arguments. `kwargs` permet de passer des arguments de mots-clés à une fonction. Si on ajoute print(kwargs), cela va retourner un dictionnaire avec tous les arguments de mot clés

Redirect

Renvoie une réponse HttpResponseRedirect à l'URL correspondant aux paramètres transmis.

Nous utilisons `redirect()` pour les fonctions d'ajouts et de retraits d'abonnements dans nos vues, en indiquant en paramètre l'URL du profil qui sera utilisée comme emplacement de redirection.

```
def post(self, request, pk, *args, **kwargs):
        profile = UserProfile.objects.get(pk=pk)
        profile.followers.add(request.user)

        return redirect('profile', profile.pk)
```

HttpResponserRedirect()

HttpResponseRedirect() prend un seul paramètre : l'URL vers laquelle l'utilisateur va être redirigé (redirige vers une nouvelle URL).

Nous l'utilisons dans nos vues pour les likes et dislikes pour rediriger l'utilisateur vers l'URL de la page où il se trouve.

```
next = request.POST.get('next','/')
return HttpResponseRedirect(next)
```

La ligne de code :

```
next = request.POST.get('next','/')
```

récupère la valeur de l'argument next dans l'objet request.POST qui contient les données soumises par l'utilisateur via le formulaire. Si next n'est pas présent dans les données soumises par l'utilisateur, la valeur par défaut "/" est utilisée.

Crsf_token

Django est livré avec une protection simple d'emploi contre les attaques de type Cross Site Request Forgeries. Lors de l'envoi d'un formulaire par la méthode POST et la protection CSRF active, nous devons obligatoirement utiliser la balise de gabarit csrf_token

```
<form method="POST">
    {% csrf_token %}
    {{ form | crispy }}
```

```
    <div class="d-grid gap-2">
        <button class="btn btn-primary mt-3">Envoyer!</button>
    </div>
</form>
```

User

Les objets utilisateurs sont au cœur du système d'authentification.

Les principaux attributs de l'utilisateur par défaut sont les suivants :

- nom d'utilisateur
- mot de passe
- email
- prenom
- nom

Timezone

```
sendingTime = models.DateTimeField(default=timezone.now)
```

Django inclut un paramètre `TIME_ZONE` dont la valeur par défaut est l'horaire 'UTC'. Il se trouve dans le fichier de configuration 'settings.py':

```
TIME_ZONE = 'Europe/Paris'
```

Nous l'avons modifié pour affiché l'horaire française

Décorateurs

```
@receiver(post_save, sender=User) #décorateur de Django
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        UserProfile.objects.create(user=instance)

@receiver(post_save, sender=User) #décorateur de Django
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()
```

Ces deux décorateurs sont utilisés pour mettre à jour automatiquement le profil utilisateur chaque fois qu'un utilisateur est créé ou sauvegardé.

Le premier décorateur, create_user_profile, est appelé après la création d'un nouvel utilisateur. Il crée automatiquement un objet UserProfile avec le nouvel utilisateur créé en tant qu'attribut "user".

Le deuxième décorateur, save_user_profile, est appelé chaque fois que l'objet utilisateur est sauvegardé. Il sauvegarde automatiquement le profil utilisateur correspondant.

Ces décorateurs sont très utiles pour automatiser les tâches de gestion de profil utilisateur, évitant ainsi les erreurs et les oublis de la part des développeurs.