Système d’authentification

Système d’authentification

•Le système d’authentification dans Django gère la connexion des utilisateurs et les autorisations affectées.

•Il permet la:

–Gestion des utilisateurs

–Gestion des groupes

–Gestion des permissions

–Système de « hachage » des mots de passe

–Des interfaces d’authentification (formulaires et vues)

–Restriction d’accès au contenu

•La classe « **User** » est la base dans le mécanisme d’authentification. (*django.contrib.auth.models.User*)

•Un objet de cette classe est caractérisé par:

–username	–is_active
–password	–is_superuser
–email	–groups
–first_name	–user_permissions
–last_name	–last_login
–is_staff	–date_joined

•La méthode directe de création d’utilisateur est « **create_user** ».

```
>>> from Django.contrib.auth.models import User
>>> user = User.objects.create_user('ali','ali@esprit.tn','mot_de_passe')
```

•L’interface d’administration de Django permet même de gérer les utilisateurs, les permissions ainsi que les groupes.

•Il est possible de vérifier si une requête web provient d’un utilisateur authentifié ou pas ainsi:

```
if request.user.is_authenticated:
    # Traitement.
else:
    # Traitement pour les utilisateurs anonymes
```

•Si un utilisateur non connecté ne détient pas le droit d'accéder à une ressource, on utilise le « shortcut » **HttpResponseForbidden** associé au Template500.html.

```
else:
    return HttpResponseForbidden()
```

Login

• ***1ère méthode***

Il est possible d'authentifier un utilisateur ainsi:

```
from django.contrib.auth import authenticate, login

def loginPage(request):
    username = request.POST['username']  password = request.POST['password']
user = authenticate(request, username=username, password=password)

if user is not None:
    if user.is_active:

        login(request, user)

        return HttpResponseRedirect(reverse('index'))
    else:
        #msg d'erreur utilisateur inactive
else:
    #msg d'erreur que l'utilisateur est inexistant
```

• ***2ème méthode***

Django fournit la classe « **LoginView** » par défaut pour faciliter la connexion des utilisateurs et la vérification des informations.

```
from django.contrib.auth.views import LoginView

urlpatterns = [
    path('login/', LoginView.as_view(), name="login"),
  …
]
```

•Dans le fichier **urls.py**, on peut passer « **template_name** » en paramètre afin de préciser le Template à utiliser.

Logout

•Il existe deux méthodes pour déconnecter un utilisateur.

1ère méthode

```
def logout_view(request):  logout(request)

    return redirect('index')
```

Remarques:

–La méthode logout ne lève pas une exception si l'utilisateur n'est pas déjà connecté.

–En appelant cette méthode, toutes les données relatives à la session d'utilisateur sont supprimées.

2ème méthode

•Django fournit la classe « **LogoutView** » par défaut pour la déconnexion des utilisateurs.

•Dans le fichier **urls.py**, on fait appel à la classe générique **LogoutView**.

```
from django.contrib.auth.views import LoginView

urlpatterns = [
    path('logout/', LogoutView.as_view(), name="logout"),
  …
]
```

•On peut passer « **next_page** » en paramètre afin de préciser la vue à afficher suite à la déconnexion.

Restriction d'accès

•Pour vérifier l'état d'un utilisateur coté « Template ».

```
{% if user.is_authenticated %}  Bienvenue {{  user.username }} !

{% else %}

Vous n’êtes pas connecté

{% endif %}
```

•Pour limiter l’accès aux vues, on vérifie si l’utilisateur est authentifié ou pas.

```
def profileView(request):
    if not user.is_authenticated:
    return redirect(‘%s?next=%s' % (settings.LOGIN_URL, request.path))
    else:
        …
```

•Pour limiter l’accès aux vues basées sur les fonctions, Django fournit des décorateurs tel que « login_required », « permission_required »…

```
from django.contrib.auth.decorators import login_required

@login_required
def profileView(request):
    return render(request, 'hub/profile.html')
```

•Pour limiter l’accès aux vues basées sur les classes, Django fournit des classes **Mixin** .

•Les Mixins permettent de faire le même travail que les décorateurs sauf que ça sera appliqué aux vues.

```
from django.contrib.auth.mixins import LoginRequiredMixin

class Profile_View(LoginRequiredMixin):
  ............
```