TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- ¿En qué consiste el modelo de permisos de Django?
- Permisos básicos.
- Grupos.
- Uso de Mixins.
- ¿Qué son los Mixin?
- Aplicando LoginRequiredMixin y PermissionRequiredMixin.
- Examinando la tabla auth_permissions.
- Redireccionando los accesos no autorizados.
- Vistas de autenticación.

AUTORIZACIÓN Y PERMISOS

Django trae integrado su propio sistema de permisos, el cual ofrece una manera de asignar la permisología necesaria a usuarios y grupos de usuarios específicos. Lo emplea el sitio de administración de Django, pero también puede usarlo en su propio código.

¿EN QUÉ CONSISTE EL MODELO DE PERMISOS DE DJANGO?

El sitio de administración de Django usa permisos de la siguiente manera:

- El acceso para ver objetos está limitado a los usuarios con el permiso de "ver" o "cambiar" para ese tipo de objeto.
- El acceso para ver el formulario "agregar" y agregar un objeto está limitado a los usuarios con el permiso "agregar" para ese tipo de objeto.
- El acceso para ver la lista de cambios, ver el formulario de "cambio", y cambiar un objeto está limitado a los usuarios con el permiso de "cambio" para ese tipo de objeto.
- El acceso para eliminar un objeto está limitado a los usuarios con el permiso de "eliminación" para ese tipo de objeto.

PERMISOS BÁSICOS

Cada vez que se crea un modelo, y se ejecutan las migraciones, se crean automáticamente 4 permisos (add, edit, delete y view) en `django.contrib.auth` para ese objeto. Cuando `django.contrib.auth` se lista en su archivo de configuración `INSTALLED_APPS`, se asegurará de que se creen cuatro permisos predeterminados (agregar, cambiar, eliminar y ver) para cada modelo de Django definido en una de sus aplicaciones instaladas.

- Estos permisos se crean al momento de ejecutar `manage.py migrate`; la primera vez que ejecute `migrate` después de agregar `django.contrib.auth` a `INSTALLED_APPS`, se crearán los permisos predeterminados para todos los modelos instalados anteriormente, así como para cualquier modelo nuevo que se instale en ese momento.
- Luego, se crearán permisos predeterminados para nuevos modelos cada vez que se ejecute `manage.py migrate`.
- En caso de que se requiera probar la permisología básica que pueda tener un usuario, se debe usar: (asumiendo que tenemos una aplicación llamada auto y un modelo llamado Ford)
    - `add: user.has_perm('auto.add_ford')`.
    - `change: user.has_perm('auto.change_ford')`.
    - `delete:user.has_perm('auto.delete_ford')`.
    - `view:user.has_perm('auto.view_ford')`.

GRUPOS

Son una forma genérica de trabajar con varios usuarios a la vez, de manera que se les pueda asignar permisos o etiquetas en bloque. Un usuario puede pertenecer a varios grupos a la vez. Un usuario que pertenezca a un grupo recibe automáticamente todos los permisos que se la hayan otorgado al grupo. Por ejemplo: si el grupo Colaboradores tiene el permiso `can_edit_home_page`, cualquier usuario que pertenezca a él también tiene ese permiso.

Los grupos también permiten categorizar a los usuarios para asignarles una determinada etiqueta, o para otorgarles una funcionalidad extra. Por ejemplo: se puede crear un grupo Usuarios especiales, y utilizar código para permitir el acceso a determinadas porciones de tu sitio sólo a los miembros de ese grupo, o para enviarles un correo electrónico sólo a ellos.

Al igual que con los usuarios, la manera más sencilla de gestionar los grupos es usando la interfaz de administración de Django.

USO DE MIXINS

Hacer uso de `mixins` puede proporcionar una gran funcionalidad, no solo para los modelos de base de datos, sino también para otras clases que pueda tener. Usarlos en modelos específicamente permite que éstos sean más fáciles y manejables. Puede escribir sus campos y métodos como `mixins`, y usarlos en cualquier modelo que desee. También hacer combinaciones para una variedad de casos de uso, como que un modelo sea auditable o concurrente, o tal vez solo necesite una funcionalidad específica para un grupo de modelos. En general, los `mixins` ayudan a mantener un código limpio. Ejemplo: si se encuentra repitiendo mucho código, es posible que desee buscar formas de abstraer esa lógica en su propia clase.

¿QUÉ SON LOS MIXIN?

Es un concepto en la Programación Orientada a Objetos que pretende resolver parte del problema de la herencia múltiple. La idea es eliminar la lógica duplicada en varias clases, y separarlas en su propia clase. A partir de ahí, se pueden "mezclar" incluir en donde se necesiten.

Básicamente, son clases que contienen objetos y propiedades que se pueden aplicar a otros objetos. Es importante considerar que los `mixins` deben contener propiedades y lógica concretas. Las clases que los usan no deben anularlos ni implementarlos. Por otra parte, importante saber que estas clases no deben valerse por sí mismas. No se deben instanciar. Se supone que proporcionan funcionalidad adicional a cualquier clase que los use.

¿PARA QUÉ NOS SIRVEN LOS MIXIN EN EL MODELO AUTH?

Una vista toma una solicitud y devuelve una respuesta. Esto puede ser más que una simple función, y Django proporciona un ejemplo de algunas clases que se pueden usar como vistas (`class-based views`). Esto permite estructurar las vistas y reutilizar el código aprovechando la herencia y los `mixins`.

APLICANDO LOGINREQUIREDMIXIN

Al utilizar vistas basadas en clases, puede lograr el mismo comportamiento que con `login_required` haciendo uso de `LoginRequiredMixin`. Este `mixin` debe estar en la posición más a la izquierda en la lista de herencia. Al hacer uso de la clase `LoginRequiredMixin` en una vista, todas las solicitudes de usuarios no autenticadas serán redirigidas a la página de inicio de sesión o se mostrará el error HTTP 403, esto dependerá del parámetro `raise_exception`. Además, se puede configurar cualquier parámetro de `AccessMixin` para el manejo de usuarios no autorizados. Ejemplo:

```
1 from django.contrib.auth.mixins import LoginRequiredMixin
2 class MyView(LoginRequiredMixin, View):
3     login_url = '/login/'
4     redirect_field_name = 'redirect_to'
```

APLICANDO PERMISSIONREQUIREDMIXIN

Para aplicar verificaciones de permisos a vistas basadas en clases, se puede utilizar `PermissionRequiredMixin`. Este `mixin`, al igual que `allow_required`, verifica si el usuario que accede a una vista tiene todos los permisos otorgados. Debe especificar el permiso (o una iteración de permisos), usando el parámetro `allow_required`:

```
1 from django.contrib.auth.mixins import PermissionRequiredMixin
2 class MyView(PermissionRequiredMixin, View):
3     permission_required = 'polls.add_choice'
4     # Or multiple of permissions:
5     permission_required = ('polls.view_choice', 'polls.change_choice')
```

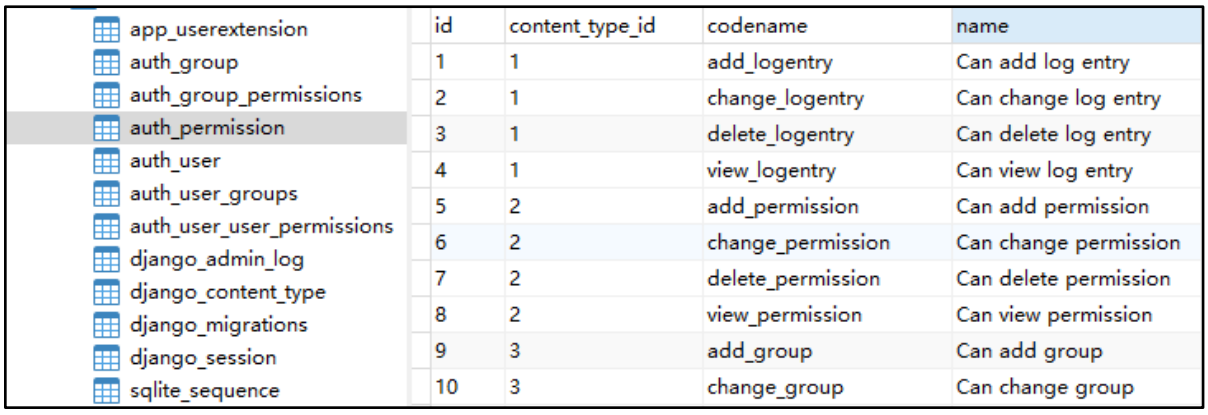
Es posible configurar cualquiera de los parámetros de `AccessMixin` para el manejo de los usuarios no autorizados. Incluso, se pueden sobrescribir los siguientes métodos: `get_permission_required()` y `has_permission()`.

- La primera devuelve una iteración de nombres de permisos usados por el `mixin`. El valor predeterminado es el atributo `permission_required`, convertido a una tupla si es necesario.
- La segunda función, devuelve un valor booleano que indica si el usuario actual tiene permiso para ejecutar la vista decorada. De forma predeterminada, devuelve el resultado de llamar a `has_perms()` con la lista de permisos devueltos por `get_permission_required()`.

EXAMINANDO LA TABLA AUTH_PERMISSIONS

Como ya hemos visto, Django cuenta con una función de permisos. Luego de crear un modelo, existen cuatro tipos de permisos para ese modelo de forma predeterminada, estos son: add (agregar), eliminar (delete), cambiar (change), y visualizar (view). Todos éstos se pueden ver en la tabla `auth_permission` de la base de datos, luego de ejecutar el comando `migrate`.

La tabla `auth_permission` cuenta con los siguientes atributos: `id`, `content_type_id`, `codename` y `name`. El atributo `id` es el identificador como tal de cada registro en la tabla; `content_type_id` representa un identificador del modelo; `codename` es el nombre en clave del permiso; y finalmente, `name` representa la función del permiso en cuestión.



id	content_type_id	codename	name
1	1	add_logentry	Can add log entry
2	1	change_logentry	Can change log entry
3	1	delete_logentry	Can delete log entry
4	1	view_logentry	Can view log entry
5	2	add_permission	Can add permission
6	2	change_permission	Can change permission
7	2	delete_permission	Can delete permission
8	2	view_permission	Can view permission
9	3	add_group	Can add group
10	3	change_group	Can change group

Si por ejemplo se crea el modelo car, Django creará cuatro permisos por defecto y los agregará a la tabla `auth_permission`: `add_car`, `change_car`, `delete_car` y `view_car`. En caso de requerir añadir un permiso adicional, por ejemplo, para editar, éste se puede especificar utilizando el atributo `permission` en la clase `Meta`, de la siguiente manera:

```
class Car(models.Model):
        model = models.CharField(max_length=100)
        year = models.TextField()
        manufacture =
models.ForeignKey(get_user_model(),on_delete=models.CASCADE)
   class Meta:
               permissions = (('edit_car','can edit car'), )
```

Los nuevos permisos definidos se crearán luego de ejecutar el comando de sincronización, y se podrán verificar en la tabla `auth_permission`.

REDIRECCIONANDO LOS ACCESOS NO AUTORIZADOS

La manera indicada de limitar el acceso a las páginas del sistema es chequear `request.user.is_authenticated`. `is_authenticated` es una forma de saber si el usuario ha sido autenticado. Esto no implica ningún permiso, y no comprueba si el usuario está activo o tiene una sesión válida. Puede ser implementada de la siguiente manera: el primer ejemplo redirecciona a la página de inicio de sesión, y el segundo despliega un mensaje de error.

```
from django.conf import settings
from django.shortcuts import redirect
def my_view(request):
    if not request.user.is_authenticated:
        return redirect('%s?next=%s' % (settings.LOGIN_URL, request.path))
```

```
from django.shortcuts import render
def my_view(request):
    if not request.user.is_authenticated:
        return render(request, 'app/login_error.html')
```

VISTAS DE AUTENTICACIÓN

Django ofrece un conjunto de vistas para el manejo de inicio de sesión (login), cierre de sesión (logout), y administración de contraseñas. Por otra parte, no existen plantillas por defecto para las vistas de autenticación, es necesario crearlas. Existen diferentes métodos para implementar estas vistas en un proyecto. La manera más fácil es incluir la `URLconf` proporcionada en `django.contrib.auth.urls` en su propia `URLconf`. Por ejemplo:

```
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Que incluyen las siguientes URL:

- `accounts/login/ [name='login']`.
- `accounts/logout/ [name='logout']`.
- `accounts/password_change/ [name='password_change']`.
- `accounts/password_change/done/ [name='password_change_done']`.
- `accounts/password_reset/ [name='password_reset']`.
- `accounts/password_reset/done/ [name='password_reset_done']`.
- `accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']`.
- `accounts/reset/done/ [name='password_reset_complete']`.

Si se desea tener más control sobre las URL, puede hacer referencia a una vista específica en su **`URLconf`**:

```
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('change-password/', auth_views.PasswordChangeView.as_view()),
]
```

Las vistas tienen argumentos opcionales que se pueden utilizar para modificar su comportamiento. Por ejemplo: si desea cambiar el nombre de la plantilla que usa una vista, puede proporcionar el argumento **`name_template`**. Una forma de hacer esto es proporcionar argumentos de palabras clave en la **`URLconf`**, estos se pasarán a la vista. De la siguiente manera:

```
urlpatterns = [
    path(
    'change-password/',
        auth_views.PasswordChangeView.as_view(template_name='change-
password.html'),
    ),
]
```

Las vistas de autenticación incluidas en **`django.contrib.auth`** son las siguientes:

- **`class LoginView`**.
- **`class LogoutView`**.
- **`class PasswordChangeView`**.

- `class PasswordChangeDoneView`.
- `class PasswordResetView`.
- `class PasswordResetDoneView`.
- `class PasswordResetConfirmView`.
- `class PasswordResetCompleteView`.

LOGINVIEW

URL: Login.

Funcionalidades:

- Si se llama a través de GET, muestra un formulario de inicio de sesión que envía POST a la misma URL.
- Si se llama a través de POST, con las credenciales enviadas por el usuario, intenta iniciar la sesión del usuario. Si el inicio de sesión es exitoso, la vista se redirige a la URL especificada a continuación. Si no se proporciona, se redirige a `configuración.LOGIN_REDIRECT_URL` (que por defecto es `/accounts/perfil/`). Si el inicio de sesión no es exitoso, vuelve a mostrar el formulario de inicio de sesión.

El HTML para la plantilla de inicio de sesión se debe proporcionar, la forma predeterminada es `registro/login.html`. A esta plantilla se le pasan cuatro variables de contexto de plantilla:

- `form`: un objeto de `Form` que representa el AuthenticationForm.
- `next`: la URL a la que se redirigirá después de un inicio de sesión exitoso. Esto también puede contener una cadena de consulta.
- `site`: el sitio actual, de acuerdo con la configuración `SITE_ID`. Si no tiene instalado el marco del sitio (framework), se establecerá en una instancia de `RequestSite`, que deriva el nombre del sitio y el dominio del `HttpRequest` actual.
- `site_name`: un alias para site.name. Si no tiene instalado el marco del sitio, se establecerá en el valor request.META['SERVER_NAME'].

Atributos y Métodos:

- `template_name`.
- `next_page`.
- `redirect_field_name`.
- `authentication_form`.
- `extra_context`.
- `redirect_authenticated_user`.
- `success_url_allowed_hosts`.
- `get_default_redirect_url()`.

LOGOUTVIEW

URL: logout

Funcionalidad: Cierre de sesión de usuario.

Atributos:

- `next_page`.
- `template_name`.
- `redirect_field_name`.
- `extra_context`.
- `success_url_allowed_hosts`.

Variables de contexto del `template`:

- `title`.
- `site`.
- `site_name`.

PASSWORDCHANGEVIEW

URL: password_change

Funcionalidad: Permite a un usuario cambiar su contraseña.

Atributos:

- `template_name`.
- `success_url`.
- `form_class`.
- `extra_context`.

Variables de contexto del `template`:

- `form`.

PASSWORDCHANGEDONEVIEW:

URL: password_change_done

Funcionalidad: Esta vista es la que se despliega luego de que un usuario ha cambiado su contraseña.

Atributos:

- `template_name`.
- `extra_context`.

PASSWORDRESETVIEW

URL: password_reset

Funcionalidad: permite al usuario reiniciar su contraseña, para lo cual envía un URL a la dirección de correo electrónico registrado por el usuario.

Las condiciones para el envío son las siguientes:

- La dirección de correo electrónico existe en la app.
- El usuario se encuentra activo en la app.
- El usuario tiene configurada una contraseña utilizable.

Atributos:

- `template_name`.

- `form_class`.
- `email_template_name`.
- `subject_template_name`.
- `token_generator`.
- `success_url`.
- `from_email`.
- `extra_context`.
- `html_email_template_name`.
- `extra_email_context`.

Variables de contexto del `template`:

- `form`.

Variables de contexto del `template` del correo electrónico:

- `email`.
- `user`.
- `site_name`.
- `domain`.
- `protocol`.
- `uid`.
- `token`.

PASSWORDRESETDONEVIEW

URL: password_reset_done

Funcionalidad: esta vista se refiere a la página que se despliega luego de que es enviado el correo electrónico al usuario que ha solicitado reiniciar su contraseña.

Atributos:

- `template_name`.

- `extra_context`.

PASSWORDRESETCONFIRMVIEW

URL: password_reset_confirm

Funcionalidad: despliega un formulario para introducir una nueva contraseña.

Atributos:

- `template_name`.
- `token_generator`.
- `post_reset_login`.
- `post_reset_login_backend`.
- `form_class`.
- `success_url`.
- `extra_context`.
- `reset_url_token`.

Variables de contexto del `template`:

- `form`.
- `validLink`.

PASSWORDRESETCOMPLETEVIEW

URL: password_reset_complete

Funcionalidad: esta vista permite informar a un usuario que su contraseña ha sido satisfactoriamente cambiada.

Atributos:

- `template_name`.
- `extra_context`.