Contenidos a Trabajar

1. Autenticación de usuarios en Django
   - Instalación
2. Usando el sistema de autenticación de Django
   - Permisos y autorización
   - Autenticación en solicitudes Web

Autenticación de usuarios en Django

Django viene con un sistema de autenticación de usuarios. Maneja cuentas de usuario, grupos, permisos y sesiones de usuario basadas en cookies. Esta sección de la documentación explica cómo funciona la implementación predeterminada, así como también cómo ampliarla y personalizarla para que se adapte a las necesidades de su proyecto.

Resumen

El sistema de autenticación de Django maneja tanto la autenticación como la autorización. Brevemente, la autenticación verifica que un usuario es quien dice ser, y la autorización determina qué puede hacer un usuario autenticado. Aquí se utiliza el término autenticación para referirse a ambas tareas.

El sistema de autenticación consta de:

- Usuarios
- Permisos: banderas binarias (sí/no) que designan si un usuario puede realizar una determinada tarea.
- Grupos: una forma genérica de aplicar etiquetas y permisos a más de un usuario.
- Un sistema de hash de contraseña configurable
- Formularios y herramientas de visualización para iniciar sesión en usuarios o restringir contenido
- Un sistema backend enchufable.

El sistema de autenticación en Django pretende ser muy genérico y no proporciona algunas características que se encuentran comúnmente en los sistemas de autenticación web. Se han implementado soluciones para algunos de estos problemas comunes en paquetes de terceros:

- Comprobación de la seguridad de la contraseña
- Limitación de los intentos de inicio de sesión
- Autenticación frente a terceros (OAuth, por ejemplo)
- Permisos a nivel de objeto

Instalación

El soporte de autenticación se incluye como un módulo de contribución de Django en **django.contrib.auth**. De forma predeterminada, la configuración requerida ya está incluida en la **settings.py** generada por '**django- admin startproject**', estos consisten en dos elementos enumerados en su configuración **INSTALLED_APPS**:

1. **'django.contrib.auth'** contiene el núcleo del framework de autenticación y sus modelospredeterminados.
2. **'django.contrib.contenttypes'** es el sistema de tipo de contenido de Django , que permite asociarpermisos con los modelos que creas.

y estos elementos en su entorno **MIDDLEWARE**:

1. **SessionMiddleware** gestiona sesiones a través de solicitudes.
2. **AuthenticationMiddleware** asocia usuarios con solicitudes mediante sesiones.

Con esta configuración en su lugar, ejecutar el comando **manage.py migrate** crea las tablas de base de datos necesarias para los modelos relacionados con la autenticación y los permisos para cualquier modelo definido en sus aplicaciones instaladas.

Usando el sistema de autenticación de Django

Este documento explica de forma muy resumida el uso del sistema de autenticación de Django en su configuración por defecto. Esta configuración ha evolucionado para satisfacer las necesidades de los proyectos más comunes, maneja una gama razonablemente amplia de tareas y tiene una implementación cuidadosa de contraseñas y permisos. Para proyectos donde las necesidades de autenticación difieren de las predeterminadas, Django admite una amplia extensión y personalización de la autenticación que debe revisar en la documentación oficial.

La autenticación de Django proporciona autenticación y autorización juntas y, en general, se la conoce como el sistema de autenticación, ya que estas funciones están unidas.

objetos **User:**

Los objetos **User** son el núcleo del sistema de autenticación. Por lo general, representan a las personas que interactúan con su sitio y se usan para habilitar cosas como restringir el acceso, registrar perfiles de usuario, asociar contenido con creadores, etc. Solo existe una clase de usuario en el marco de autenticación de Django, es decir, los usuarios **'superusers'** administradores son solo objetos de usuario con conjunto de atributos especiales **'staff'**, no diferentes clases de objetos de usuario.

Los atributos principales del usuario predeterminado son:

- **username**
- **password**
- **email**
- **first_name**
- **last_name**

Consulte la documentación oficial para obtener una referencia completa, lo que continúa está más orientado a tareas y resumir conceptos básicos. La creación de usuarios, grupos y manejo de los mismos, se demuestra con objetos por consola para entender su funcionamiento, pero se recomienda en la práctica utilizar el admin de django para su manejo.

Creando usuarios

La forma más directa de crear usuarios es usar la función auxiliar incluida **create_user()**:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john', 'lennon@thebeatles.com',
'johnpassword')

# At this point, user is a User object that has already been saved # to the
database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = 'Lennon'
>>> user.save()
```

El superusuario se crea utilizando el manage.py por consola.

Cambio de contraseñas

Django no almacena contraseñas sin procesar (texto claro) en el modelo de usuario, sino solo un hash (consulte la documentación sobre cómo se administran las contraseñas para obtener detalles completos). Por este motivo, no intente manipular directamente el atributo de contraseña del usuario. Esta es la razón por la que se utiliza una función auxiliar al crear un usuario.

Para cambiar la contraseña de un usuario, tienes varias opciones:

manage.py changepassword *username* ofrece un método para cambiar la contraseña de un usuario desde la línea de comandos. Le solicita que cambie la contraseña de un usuario determinado que debe ingresar dos veces. Si ambos coinciden, la nueva contraseña se cambiará inmediatamente. Si no proporciona un usuario, el comando intentará cambiar la contraseña cuyo nombre de usuario coincida con el usuario actual del sistema.

También puede cambiar una contraseña mediante programación, usando **set_password()**:

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

Autenticando usuarios

authenticate(*solicitud=Ninguna , **credenciales*)

Use **authenticate()** para verificar un conjunto de credenciales. Toma las credenciales como argumentos de palabras clave **username** y , **password** en el caso predeterminado, las compara con cada backend de autenticación y devuelve un **User** objeto si las credenciales son válidas para un backend. Si las credenciales no son válidas para ningún backend o si un backend genera **PermissionDenied**, devuelve **None**. Por ejemplo:

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # A backend authenticated the credentials
else:
    # No backend authenticated the credentials
```

Permisos y autorización

Django viene con un sistema de permisos integrado. Proporciona una forma de asignar permisos a usuarios ygrupos de usuarios específicos.

Lo usa el sitio de administración de Django, pero puede usarlo en su propio código.El sitio de administración de Django usa permisos de la siguiente manera:

- El acceso a la visualización de objetos está limitado a usuarios con el permiso «ver» o «cambiar» paraese tipo de objeto.
- El acceso para ver el formulario «agregar» y agregar un objeto está limitado a los usuarios con elpermiso «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 «eliminar» para ese tipode objeto.

Los permisos se pueden establecer no solo por tipo de objeto, sino también por instancia de objeto específica. Mediante el uso de los métodos , y proporcionados por los métodos **has_view_permission()**, **has_add_permission()**, **has_change_permission()**, **has_delete_permission()** por la clase **ModelAdmin**, es

posible personalizar los permisos para diferentes instancias de objetos del mismo tipo.

Los objetos **User** tienen dos campos de muchos a muchos: **groups** y **user_permissions**. Los objetos **User** pueden acceder a sus objetos relacionados de la misma manera que cualquier otro modelo de Django :

```
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

Para más información sobre los permisos generados por defecto, y como configurar dichos permisos desde el código (por más que es algo que no se hará frecuentemente), por favor revise la documentación oficial en https://docs.djangoproject.com/es/3.2/topics/auth/default/#permissions-and-authorization

Autenticación en solicitudes Web

Django usa sesiones y middleware para conectar el sistema de autenticación a objetos **request.**

Estos proporcionan un atributo **request.user** en cada solicitud que representa al usuario actual. Si el usuario actual no ha iniciado sesión, este atributo se establecerá en una instancia de **AnonymousUser**, de lo contrario, será una instancia de **User**.

Para distinguirlos el objeto user tiene un atributo booleano **is_authenticated**.

Cómo iniciar sesión en un usuario

Si tiene un usuario autenticado que desea adjuntar a la sesión actual, esto se hace con una función **login()**.**login(*solicitud* , *usuario* , *backend=Ninguno*)**

Para iniciar la sesión de un usuario, desde una vista, use **login()**. Se necesita un objeto **HttpRequest** y un objeto **User**. **login()** guarda la ID del usuario en la sesión, utilizando el marco de trabajo de sesión de Django.

Tenga en cuenta que cualquier conjunto de datos durante la sesión anónima se retiene en la sesión despuésde que un usuario inicia sesión.

Cómo cerrar la sesión de un usuario

logout(*solicitud*)

Para cerrar la sesión de un usuario que ha iniciado sesión a través de **django.contrib.auth.login()**,use **django.contrib.auth.logout()** dentro de su vista.

Tenga en cuenta que **logout()** no arroja ningún error si el usuario no inició sesión.

Cuando llama a **logout()**, los datos de la sesión para la solicitud actual se limpian por completo. Se eliminan todos los datos existentes. Esto es para evitar que otra persona use el mismo navegador web para iniciar sesión y tener acceso a los datos de la sesión del usuario anterior. Si desea agregar algo a la sesión que estará disponible para el usuario inmediatamente después de cerrar la sesión, hágalo *después de* llamar al **django.contrib.auth.logout()**.

Limitar el acceso a usuarios registrados

Una vez definidos usuarios y permisos, pude limitar el acceso a estas a vistas y templates, ya sea validando si están autenticados, o si tienen un permiso específico para acceder a determinada sección. Para eso podrá utilizar las propiedades y métodos del objeto user, como por ejemplo is_authenticated, así como también los decoradores específicos. A su vez, debido a que Django promueve el desarrollo orientado a objetos mediante clases, también tendrá Vistas Basadas en Clases que se encargar específicamente de controlar la autenticación y autorización y que podrá combinar como se vio con los Mixins en la sección de vistas basadas en clases.

A su vez, utilizando el Django template languaje, podrá a través de los objetos Request utilizados, indicarle a los templates que mostrar basándose si el usuario está autenticado o autorizado para realizar determinada acción. Para todo esto recomendamos que vea en detalle la sección de la documentación donde detalla su manejo: https://docs.djangoproject.com/es/3.2/topics/auth/default/#limiting-access-to-logged-in-users