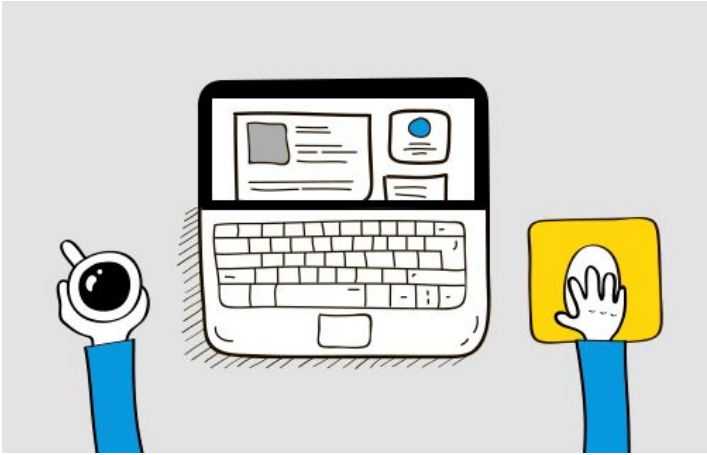
Sesiones, Autenticación y Permisos

Jorge Agustín Barón Abad

Sesiones

Es un elemento de la página Web, que almacena, registra y monitoriza las acciones de un usuario en la Web.

Cuando un usuario accede una página Web, las acciones que realiza suelen estar guardadas por un objeto que controla dicha acción del usuario en la página Web, de esta forma aunque el usuario pase de una página a otra, esa información queda almacenada.



Sesiones

¿Para qué nos servirá en nuestra aplicación Web?

La usaremos con Django para controlar el usuario logueado en Django, para gestionar un histórico del usuario que realiza las acciones en nuestra aplicación.

Sesiones

Para activarlo en Django, debemos incluir lo siguiente en el archivo **mysite/settings.py:**

```
INSTALLED_APPS = [
    ...
    'django.contrib.auth',
    'django.contrib.contenttypes',
 ....
]
```

```
MIDDLEWARE = [
    ...
    ''django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
   ...
]
```

Sesiones

Para probar que funciona. Vamos a guardar en la sesión una variable que indica a la hora que entre por primera vez en la Web y que me muestre en la cabecera siempre esa información:

En la view que muestra el index, incluimos lo siguiente:

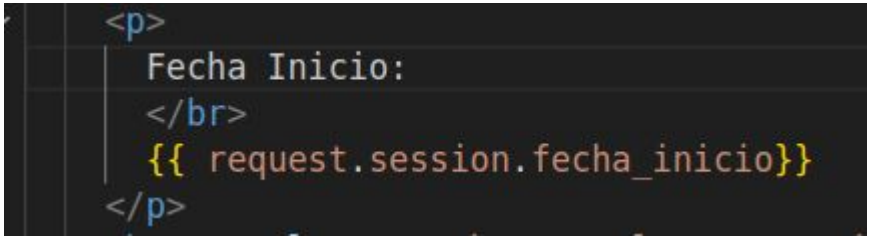
```
def index(request):

    if(not "fecha_inicio" in request.session):
        request.session["fecha_inicio"] = datetime.now().strftime('%d/%m/%Y %H:%M')

    return render(request, 'index.html')
```

¿Qué entendemos por este código?

- Las sesiones están guardadas en un atributo del objeto **request** y es un diccionario de clave- valor.
- En el código comprobamos que no se haya guardado ya un valor en esa sesión y lo guardamos por primera vez

Sesiones

Para mostrar después ese valor en las Templates, sólo debemos hacer lo siguiente:



No es necesario pasar por parámetro la variable ni nada, sólo tenemos que usar el objeto **request** y su atributo **session.**

Si incluimos este código en un template que se usa en todas las páginas, como el menú, podréis comprobar que ese valor aparecerá en todas las páginas, sin necesidad de volver asignar ese valor en **request.session[“fecha_inicio”]**

Sesiones

¿Pero ,cuándo desaparece ese valor?

Ese valor solo desaparecerá cuando el navegador lo elimine o desde la aplicación le indiquemos que lo elimine.

Para eliminarla desde Django, hacemos lo siguiente:

```
del request.session['fecha_inicio']
```

Autenticación

Ahora que entendemos que es una sesión. Vamos a explicar como añadir la autenticación a nuestra aplicación Web, que no es más que en la sesión guardemos el usuario que usa la aplicación.

Además ahora debemos empezar a controlar qué puede hacer nuestros usuarios de la Web, cuando están logueados y cuando no lo están.

Y por último veremos, que una vez logueado, que permisos tienen en la aplicación, y que podrán hacer y que no podrán hacer, dependiendo de los permisos que tenga.

Autenticación

Lo primero es definir cuál va a ser nuestro usuario en la aplicación en nuestro modelo, y cambiar la clase que hereda **models.Model** a **AbstractUser:**

```
from django.contrib.auth.models import AbstractUser
```

¿Qué destacamos?

Los campos de nombre, apellido, email, password, ya no son necesarios incluirlos, porque ya lo utilizamos de la clase padre. Nosotros sólo vamos a incluir los diferentes tipos de usuario que existirán.

En este ejemplo definiremos por defetcto la clase administrador. Pero no es lo más seguro, pero por sencillez y comodidad lo haremos así. Y más adelante veremos cómo hacerlo correctamente

```
class Usuario(AbstractUser):
    ADMINISTRADOR = 1
    CLIENTE = 2
    BIBLIOTECARIO = 3
    ROLES = (
        (ADMINISTRADOR, 'administardor'),
        (CLIENTE, 'cliente'),
        (BIBLIOTECARIO, 'bibliotecario'),
    )

    rol  = models.PositiveSmallIntegerField(
        choices=ROLES,default=1
    )
```

Autenticación

Una vez creado el modelo que hereda de **AbstractUser**, debemos indicar en la configuración de Django en **mysite/settings.py** lo siguiente:

```
AUTH_USER_MODEL = 'biblioteca.Usuario'
```

Debemos indicarle cual es el modelo que se autentifica en la plataforma. Este paso hay que hacerlo antes de realizar las migraciones

Autenticación

A continuación debemos definir cada uno de nuestros tipos de usuarios como modelos y relacionarlos con este modelo de usuario con una relación **OneToOne** que es quién se autentifica en la aplicación:

```
class Bibliotecario(models.Model):
    usuario = models.OneToOneField(Usuario,
                                   on_delete = models.CASCADE)
```

```
class Cliente(models.Model):
    usuario = models.OneToOneField(Usuario,
                                   on_delete = models.CASCADE)
    puntos = models.FloatField(default=5.0,db_column = "puntos_biblioteca")
    libros = models.ManyToManyField(Libro, through='Prestamo',related_name='prestamos_libros')
```

Autenticación

Para ver todos los campos heredados de user, lo más sencillo es crear la migraciones y ver el archivo de migraciones. Si os da error al hacer migrate, eliminar todas las migraciones y la base de datos.

```
('id', models.BigAutoField(auto_created=True, prima
('password', models.CharField(max_length=128, verbo
('last_login', models.DateTimeField(blank=True, nul
('is_superuser', models.BooleanField(default=False,
('username', models.CharField(error_messages={'unic
('first_name', models.CharField(blank=True, max_len
('last_name', models.CharField(blank=True, max_leng
('email', models.EmailField(blank=True, max_length=
('is_staff', models.BooleanField(default=False, hel
('is_active', models.BooleanField(default=True, hel
('date_joined', models.DateTimeField(default=django
('rol', models.PositiveSmallIntegerField(choices=[(
('groups', models.ManyToManyField(blank=True, help_
('user_permissions', models.ManyToManyField(blank=T
```

Autenticación

¿Qué es lo primero que debemos hacer?

Crear el formulario de registro para que los usuarios puedan registrarse y usar la aplicación.

Creamos el formulario en **myapp/forms**.py. La única diferencia con otros formularios que ya hemos visto es que hereda del formulario de crear usuario. IMPORTANTE, definimos el campo ROL, con solo los usuarios que consideremos oportunos que se registren:

```
from django.contrib.auth.forms import UserCreationForm
```

```
class RegistroForm(UserCreationForm):
    roles = (
                                (Usuario.CLIENTE, 'cliente'),
                                (Usuario.BIBLIOTECARIO, 'bibliotecario'),
            )
    rol = forms.ChoiceField(choices=roles)
    class Meta:
        model = Usuario
        fields = ('username', 'email', 'password1', 'password2','rol')
```

Autenticación

Ahora debemos crear la url y vista correspondiente:

```
path('registrar',views.registrar_usuario,name='registrar_usuario'),
```

¿Qué destacamos?

Dependiendo del rol que seleccionemos, crearemos un modelo u otro.

```
def registrar_usuario(request):
    if request.method == 'POST':
        formulario = RegistroForm(request.POST)
        if formulario.is_valid():
            user = formulario.save()
            rol = int(formulario.cleaned_data.get('rol'))
            if(rol == Usuario.CLIENTE):
                cliente = Cliente.objects.create( usuario = user)
                cliente.save()
            elif(rol == Usuario.BIBLIOTECARIO):
                bibliotecario = Bibliotecario.objects.create(usuario = user)
                bibliotecario.save()
    else:
        formulario = RegistroForm()
    return render(request, 'registration/signup.html', {'formulario': formulario})
```

Autenticación

Ahora crearemos una carpeta en **templates** que se llamara **registration** y dentro de esta carpeta el formulario la la plantilla **signup.html**:

```
{% extends "estructura/principal.html" %}

{% block contenido %}
<form action = "{% url 'registrar_usuario' %}" method = "post">

    {% load django_bootstrap5 %}
    {% csrf_token %}
    {% bootstrap_form formulario %}

    {% bootstrap_button "Enviar" button_type="submit" button_class="btn-primary" %}
</form>
{% endblock %}
```

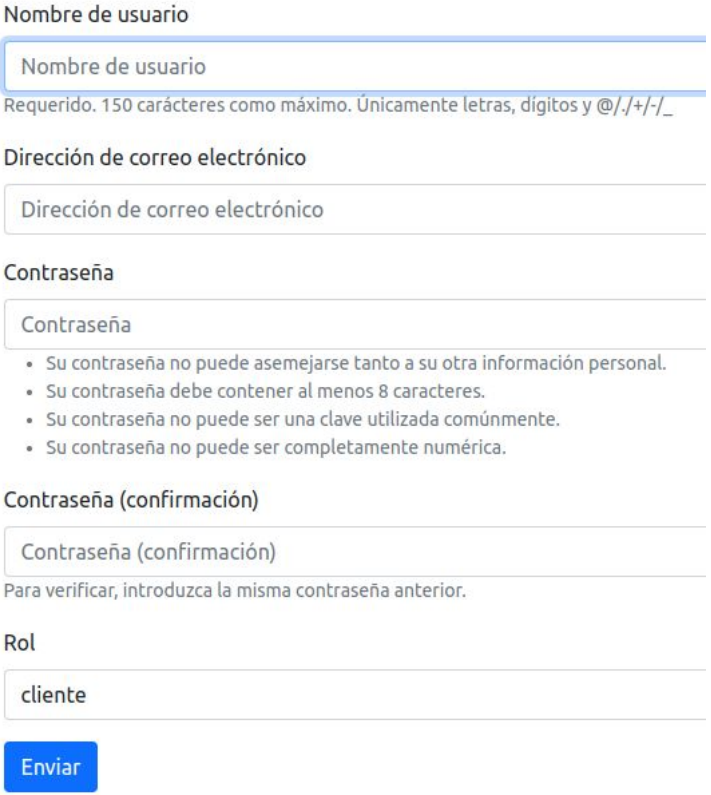
Autenticación

Por último creamos el enlace en el menu, para que el usuario pueda acceder a dicha página de registro:

```
<a class="btn btn-outline-warning" href="{% url 'registrar_usuario' %}">
    Registrarse
</a>
```

Autenticación

El resultado de este código sería el siguiente en nuestra página Web:



Autenticación

El siguiente paso es reutilizar todas las urls que ofrece Django para gestionar el login y logout de un usuario.

En el archivo **mysite/urls.py** incluimos lo siguiente:

```
path('accounts/', include('django.contrib.auth.urls')),
```

También es importante definir a qué página iremos cuando hacemos login y cuando logout. en el archivo **mysite/settings.py** indicamos lo siguiente. En mi caso la url es **index,** para ambas acciones:

```
LOGIN_REDIRECT_URL = 'index'
LOGOUT_REDIRECT_URL = 'index'
```

Autenticación

Ahora creamos en el menu las opciones de login y logout, pero vamos a añadir un condicionante a estos enlaces para dar mayor coherencia a la página.

- ¿Tiene sentido ver el botón de registrarse si ya estoy logueado?
- ¿Tiene sentido ver el botón de logout si estoy ya logueado?

Respondo a estas preguntas en la siguiente diapositiva incluyendo el código de los enlaces del menu

Autenticación

Usaremos siempre la misma condición para saber si un usuario está logueado o no, y mostrar un botón u otro-

```
{% if request.user.is_authenticated %}
```

```
{% if not request.user.is_authenticated %}
```

```
<div class="ml-2">
  {% if not request.user.is_authenticated %}
  <a class="btn btn-outline-warning" href="{% url 'login' %}">
      Login
  </a>
  {% endif %}

  {% if not request.user.is_authenticated %}
    <a class="btn btn-outline-warning" href="{% url 'registrar_usuario' %}">
        Registrarse
    </a>
   {% endif %}

   {% if request.user.is_authenticated %}
   <form action="{% url 'logout' %}" class="d-flex m-3" role="search" method="POST">
    {% csrf_token %}
    <button class="btn btn-light btn-outline-dark" type="submit">  Desconectarse</button>
  </form>
  {% endif %}
```

Autenticación

Por último nos queda incluir el formulario de login en la carpeta **registration,** el nombre de la template debe llamarse **login.html.**

```
{% extends "estructura/principal.html" %}

{% block contenido %}
<form action = "{% url 'login' %}" method = "post">

    {% load django_bootstrap5 %}
    {% csrf_token %}
    {% bootstrap_form form %}

    {% bootstrap_button "Enviar" button_type="submit" button_class="btn-primary" %}
</form>
{% endblock %}
```

Autenticación

SI pulsamos en el botón de login, nos debería aparecer una página como la siguiente:

Autenticación

Por último, para mejorar el registro, y permitir que un usuario se acaba de registrar, se logue directamente en la página, realizaremos lo siguiente:

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

```
def registrar_usuario(request):
    if request.method == 'POST':
        formulario = RegistroForm(request.POST)
        if formulario.is_valid():
            user = formulario.save()
            rol = int(formulario.cleaned_data.get('rol'))
            if(rol == Usuario.CLIENTE):
                cliente = Cliente.objects.create( usuario = user)
                cliente.save()
            elif(rol == Usuario.BIBLIOTECARIO):
                bibliotecario = Bibliotecario.objects.create(usuario = user)
                bibliotecario.save()

            login(request, user)
            return redirect('index')
    else:
        formulario = RegistroForm()
    return render(request, 'registration/signup.html', {'formulario': formulario})
```

Usamos la función **login** y pasamos por parámetros la petición(request) y el usuario que acabamos de crear(user), y hacemos un return a la página de inicio(index).

Sesión Usuario

Ahora que el usuario que gestionamos el usuario que accede, debemos tener en cuenta lo siguiente:

- Cuando creamos un modelo que está asociado al usuario, el id del usuario si usamos Formulario de Modelos, debemos crearlo como campo oculto y con el valor inicial que provenga del request.user

```
formulario = PrestamoForm(initial={"cliente":request.user.cliente})
```

```
class PrestamoForm(ModelForm):
    class Meta:
        model = Prestamo
        fields = ('libro','cliente')
        widgets = {
            "cliente":forms.HiddenInput()
        }
```

¿Qué destacamos?

Indicamos el valor inicial del usuario al crear el formulario y luego lo indicamos con un campo oculto.

Sesión Usuario

También podemos utilizar un formulario genérico.

```python
class PrestamoFormGenerico(forms.Form):
    #Definimos un campo Select para selecci
    librosdisponibles = Libro.objects.all()
    libro = forms.ModelChoiceField(
            queryset=librosdisponibles,
            widget=forms.Select,
            required=True,
            empty_label="Ninguna"
    )
```

```python
def prestamo_crear_generico(request):
    if request.method == 'POST':
        formulario = PrestamoFormGenerico(request.POST)
        if formulario.is_valid():
            try:
                prestamo = Prestamo.objects.create(
                    libro = formulario.cleaned_data.get('libro'),
                    cliente = request.user.cliente,
                )
                prestamo.save()
                return redirect("prestamo_lista_usuario",usuario_id=request.user.cliente.id)
            except Exception as error:
                print(error)
    else:
        formulario = PrestamoFormGenerico()
    return render(request, 'prestamo/create.html', {'formulario': formulario})
```

En este caso no es necesario pasar el campo como oculto porque lo vamos a guardar cuando creemos el préstamo.
Solo tenemos que indicar el libro en el formulario

Sesión Usuario

Por último tenemos que destacar lo siguiente. ¿Qué pasaría si yo al crear el formulario me gustaría que me mostrase aquellos libros que no tengo ya préstamos o que al validar me controle no pedir un préstamo del mismo libro ?

Este caso es un poco más complejo, porque debemos modificar el formulario para que acepte el parámetro de request, cuando creamos el usuario.

La forma del formulario va a cambiar un poco, pero es la única forma que nos permite Django para realizar esta tarea

Sesión Usuario

```python
class PrestamoFormGenericoRequest(forms.Form):

    def __init__(self, *args, **kwargs):
        self.request = kwargs.pop("request")
        super(PrestamoFormGenericoRequest, self).__init__(*args, **kwargs)
        librosdisponibles = Libro.objects.exclude(prestamo__cliente=self.request.user.cliente).all()
        self.fields["libro"] = forms.ModelChoiceField(
            queryset=librosdisponibles,
            widget=forms.Select,
            required=True,
            empty_label="Ninguna"
        )
```

¿Qué destacamos?

Sobreescribimos la función init, para permitir que acepte el request.

Los campos del formulario se guardan en el atributo **self.fields**

Permisos

Una vez que sabemos gestionar el usuario que entra en nuestra aplicación y las acciones que realiza. Vamos a controlar sus permisos.

¿Qué significa eso?

Es posible que dependiendo del usuario, uno realice una funcionalidad en la aplicación y otro realice otras.

Por ejemplo, el administrador podrá crear libros.

Pero un cliente no debería tener permisos para crear libros pero sí de realizar prestamos.

Permisos

Cada vez que definimos un modelo. Django crea 4 permisos asociados al modelo. Los permisos son los siguientes. Por ejemplo en nuestra aplicación biblioteca y el modelo Libro, se crearían los permisos:

biblioteca.add_libro

biblioteca.change_libro

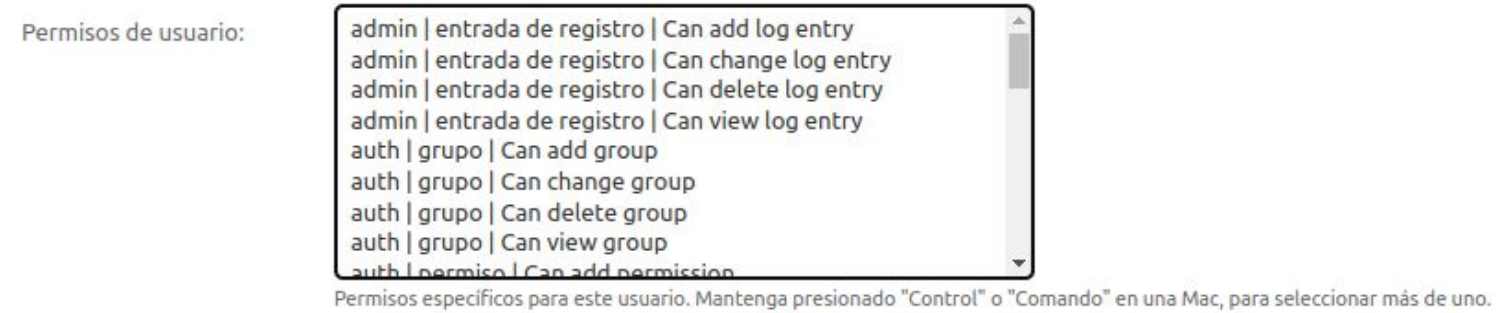
biblioteca.delete_libro

biblioteca.view_libro

Permisos

Por defecto los usuarios no tienen permisos ninguno

Para comprobarlo sólo tendremos que acceder al panel de administración y ver los permisos que tiene asociado al editar un usuario



Permisos

Después para controlar que vista puede acceder un usuario u otro,sólo debemos poner la condición de la siguiente forma.

```
from django.contrib.auth import login
from django.contrib.auth.decorators import permission_required
```

```
@permission_required('biblioteca.add_libro')
def libro_create(request):
```

Al añadir este permiso a la vista, todos los usuarios que accedan que no tengan ese permiso, no podrán entrar en esa vista y te redireccionará directamente a la página de login

Permisos

En las templates, para controlar si tiene permiso para una vista, podremos usar la siguiente línea de código:

```
{% if perms.biblioteca.add_libro %}
    <li><a class="dropdown-item" href="{% url 'libro_create' %}">Crear Libro</a></li>
{% endif %}
```

Permisos

Ahora que sabemos cómo van los permisos. ¿Cómo hacemos que un usuario tenga unos permisos u otros?

Para ello usaremos los grupos de permisos.

Un grupo de permisos permite definir varios permisos para un usuario en concreto y después asignamos ese usuario al grupo correspondiente.

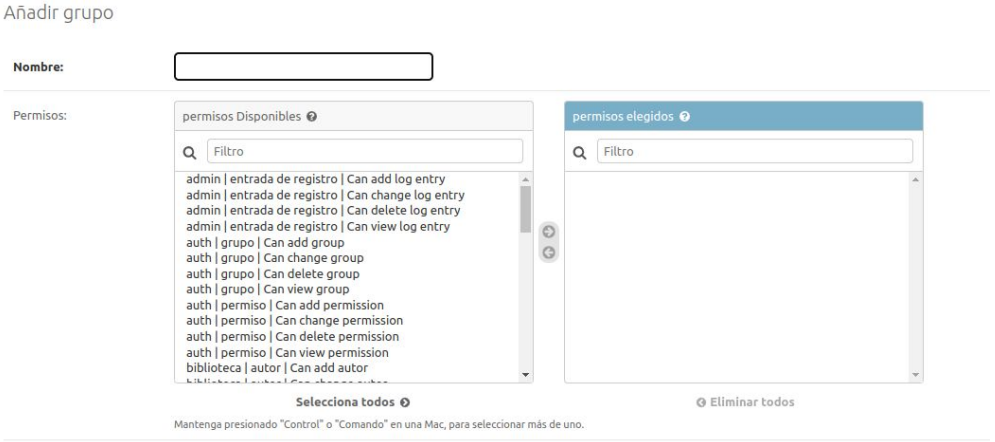
Estos grupos, se crearán desde el panel de administración y sólo podrá crearlo el usuario que tiene permisos para todo por defecto: **admin** o el que creamos con el comando **createsuperadmin**

Grupos

Vamos a crear dos grupos, uno para los bibliotecarios y otro para los clientes.

Los bibliotecarios podrán crear, editar y borrar libros. Y los clientes podrán crear préstamos.

Los grupos los llamaremos bibliotecarios y clientes respectivamente.



Grupos

¿Cömo asignamos esos grupos?

Desde el panel de administración, podemos hacerlo. Pero lo conveniente sería que al crear cada usuario se le asignase el grupo automáticamente.

Para ello debemos cambiar el código de registro y al crear un usuario Cliente o Bibliotecario asignar dichos permisos.

Grupos

```python
def registrar_usuario(request):
    if request.method == 'POST':
        formulario = RegistroForm(request.POST)
        if formulario.is_valid():
            user = formulario.save()
            rol = int(formulario.cleaned_data.get('rol'))
            if(rol == Usuario.CLIENTE):
                grupo = Group.objects.get(name='Clientes')
                grupo.user_set.add(user)
                cliente = Cliente.objects.create( usuario = user)
                cliente.save()
            elif(rol == Usuario.BIBLIOTECARIO):
                grupo = Group.objects.get(name='Bibliotecarios')
                grupo.user_set.add(user)
                bibliotecario = Bibliotecario.objects.create(usuario = user)
                bibliotecario.save()

            login(request, user)
            return redirect('index')
    else:
        formulario = RegistroForm()
    return render(request, 'registration/signup.html', {'formulario': formulario})
```

Grupos

De esta forma tendríamos los permisos correctamente asignados a cada tipo de Usuario.

Si creamos nuevos tipos de Usuario, deberíamos añadir nuevos roles y crear nuevos grupos con nuevos permisos.