Clase 23. PYTHON

Playground Avanzado Parte II

Esta clase va a ser

- grabada

Temario

22	**23**	24
Playground avanzado Parte I	**Playground avanzado Parte II**	Playground Avanzado Parte III
✓ CRUD	✓ Login - Registro - Logout	✓ Edición de usuario
✓ Clases basadas en vistas	✓ Mixin y Decoradores	✓ Avatar
		✓ Unit test

CODERHOUSE

Objetivos de la clase

- **Generar** un Login sin el panel de administración.
- **Realizar** el registro en nuestra web.
- **Inducir** a nuestro público a loguearse para ver nuestra web.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Login – Registro – Logout

Login y Logout

Hemos logrado hacer **CRUD** con Profesores y con Cursos, profesores a mano y Cursos usando **ListView** y sus derivados.
Podríamos hacer lo mismo con todas las clases, pero....
¿Es adecuado que cualquiera con acceso a nuestra web pueda hacer ABM de nuestros modelos?

Login y Logout

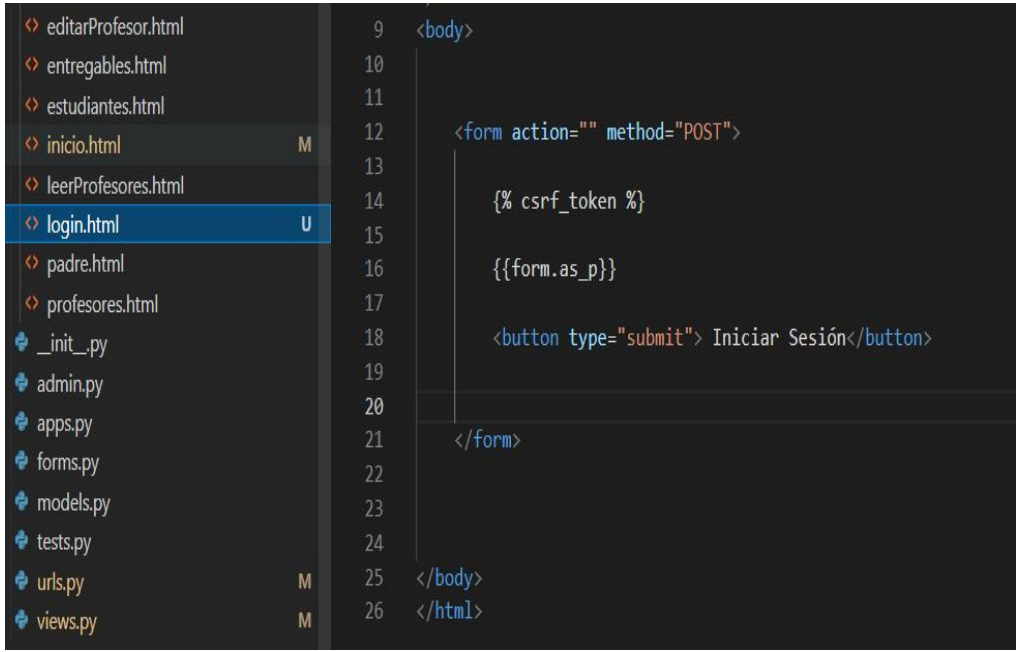
¡CLARO QUE NO!

👉 **Debemos limitar los permisos de los usuarios identificándolos con un login, similar al panel de admin.**

CODERHOUSE

Login

Login



Django nos provee de un Login y Logout muy sencillo, seguro y fácil de comprender.

👈 Arranquemos primero con un template simple para el login, login.html (form, automático de Django).

Login – Script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="" method="POST">
        {% csrf_token %}
            {{ form.as_p }}
        <button type="submit">Inciar sesion</button>
    </form>
</body>
</html>
```

Login

```
path('login', views.login_request, name = 'Login'),
```

👈 Agregamos las urls.

```
path('login', views.login_request,
name="Login")
```

```
#Para el login

from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, logout, authenticate
```

👈 Y en las views agregamos lo que necesitamos importar.

```
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth import login, logout, authenticate
```

CODERHOUSE

Login

```python
def login_request(request):


    if request.method == "POST":
        form = AuthenticationForm(request, data = request.POST)

        if form.is_valid():
            usuario = form.cleaned_data.get('username')
            contra = form.cleaned_data.get('password')

            user = authenticate(username=usuario, password=contra)


            if user is not None:
                login(request, user)

                return render(request,"AppCoder/inicio.html",  {"mensaje":f"Bienvenido {usuario}"} )
            else:

                return render(request,"AppCoder/inicio.html", {"mensaje":"Error, datos incorrectos"} )

        else:

            return render(request,"AppCoder/inicio.html" ,  {"mensaje":"Error, formulario erroneo"})

    form = AuthenticationForm()

    return render(request,"AppCoder/login.html", {'form':form} )
```

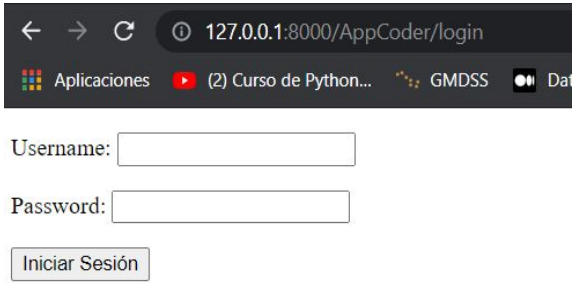
👈 La vista

Ver script: Vista.txt

CODERHOUSE

Login

Resultados ✋



Datos correctos

Datos incorrectos

CODERHOUSE

Registro

Registro

```python
def register(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)
        #form = UserRegisterForm(request.POST)
        if form.is_valid():

            username = form.cleaned_data['username']
            form.save()
            return render(request,"AppCoder/inicio.html" ,  {"mensaje":"Usuario Creado :)"})


    else:
        form = UserCreationForm()
        #form = UserRegisterForm()

    return render(request,"AppCoder/registro.html" ,  {"form":form})
```

Lo anterior funciona si tenemos algún usuario creado por panel de administración, pero también podríamos registrar usuarios sin el panel.

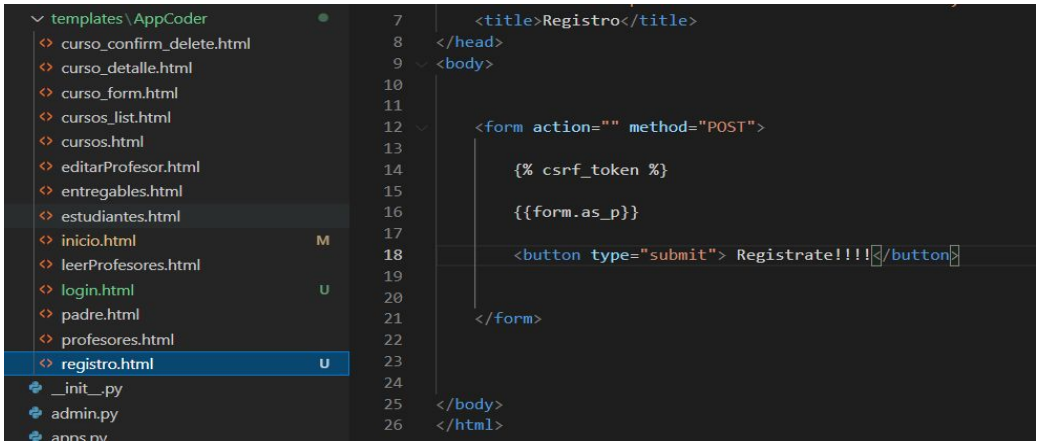
👉Veamos la vista.

Ver: Registro.txt

Registro

```
path('login', views.login_request, name = 'Login'),
path('register', views.register, name = 'Register'),
```

```
path('register', views.register,
name='Register'),
```



Modificamos las urls y creamos el template

👈

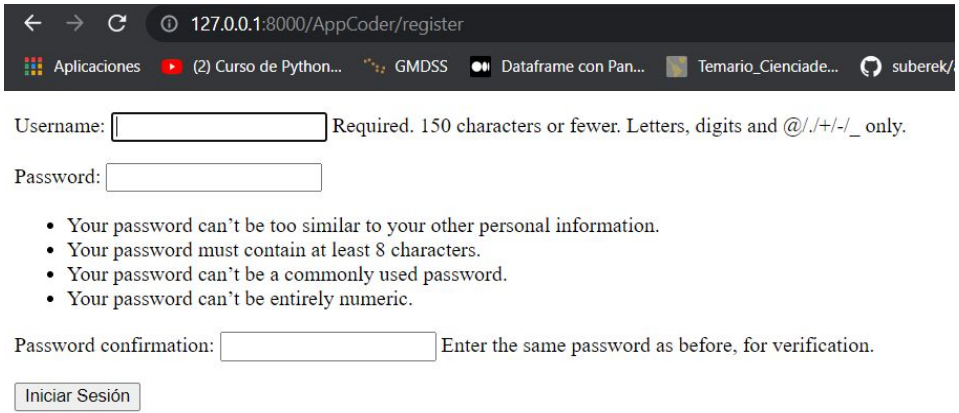
CODERHOUSE

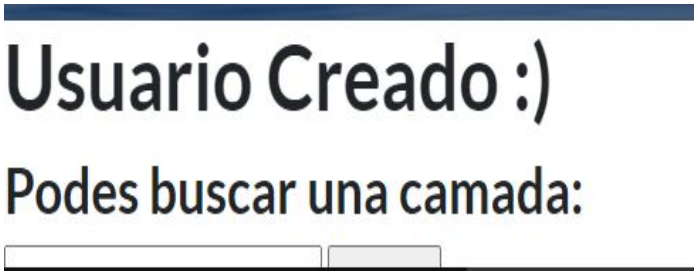
Script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="" method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Registrate!</button>
    </form>
</body>
</html>
```

Registro

Nuestro resultado se verá así 😎

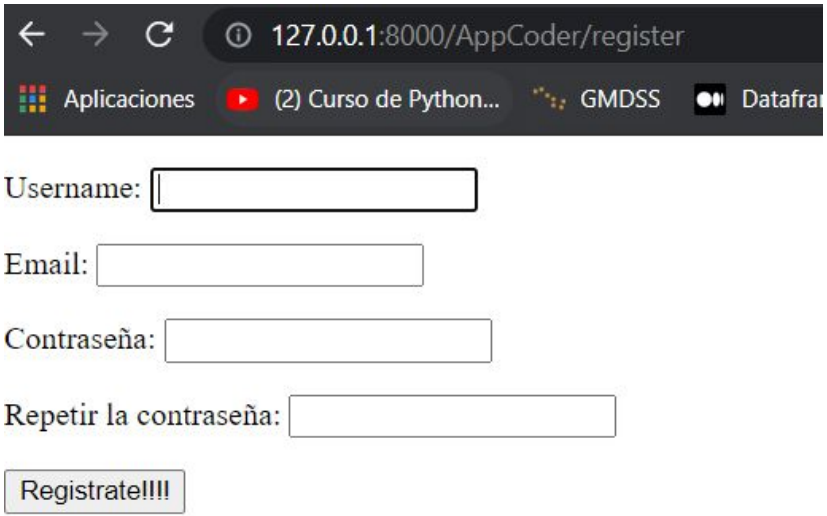




CODERHOUSE

Registro

Funcionó, pero por defecto no se ve muy estético 😞



Lo arreglaremos para que se vea así 👉

CODERHOUSE

Registro

Para eso necesitamos heredar el formulario de Django y modificarlo a nuestro gusto, desde forms.py y luego usar “nuestro” form 👇

```python
class UserRegisterForm(UserCreationForm):

    email = forms.EmailField()
    password1 = forms.CharField(label='Contraseña', widget=forms.PasswordInput)
    password2 = forms.CharField(label='Repetir la contraseña', widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
        #Saca los mensajes de ayuda
        help_texts = {k:"" for k in fields}
```

Registro

Script:

```
class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()
    password1 = forms.CharField(label="Contraseña", widget=forms.PasswordInput)
    password2 = forms.CharField(label="Repetir contraseña", widget=forms.PasswordInput)


    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
        # Saca los mensajes de ayuda
        help_texts = {k:"" for k in fields}
```

Registro

Y en la vista usamos “nuestro” form 👇

```python
def register(request):

    if request.method == 'POST':

        #form = UserCreationForm(request.POST)
        form = UserRegisterForm(request.POST)
        if form.is_valid():

            username = form.cleaned_data['username']
            form.save()
            return render(request,"AppCoder/inicio.html" ,  {"mensaje":"Usuario Creado :)"})


    else:
        #form = UserCreationForm()
        form = UserRegisterForm()

    return render(request,"AppCoder/registro.html" ,  {"form":form})
```

Logout

Logout

Django también tiene vistas listas para login y logout, usemos una vista predefinida para el Logout, en urls.py

👉

```
#cosas para el login
from django.contrib.auth.views import LogoutView
```

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

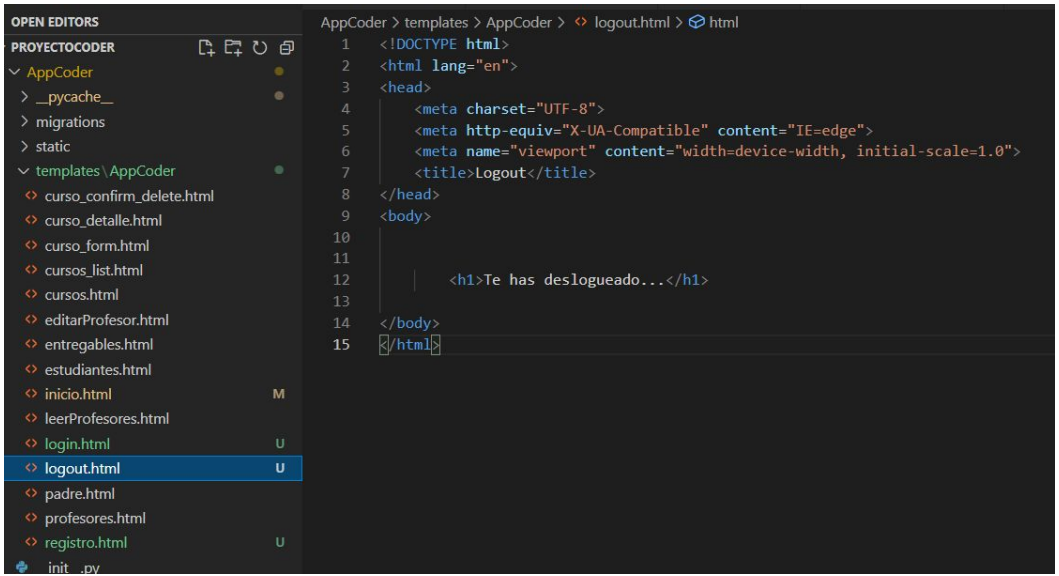
👉

```
path('login', views.login_request, name = 'Login'),
path('register', views.register, name = 'Register'),
path('logout', LogoutView.as_view(template_name='AppCoder/logout.html'), name = 'Logout'),
```

```
path('logout', LogoutView.as_view(template_name='AppCoder/logout.html'), name='Logout'),
```

CODERHOUSE

Logout



Creamos el logout.html 😮

CODERHOUSE

Logout

Script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <h1>Te has deslogueado...</h1>

</body>
</html>
```

Para pensar

Supongamos que ahora queremos que solo personas con login puedan ver nuestra página web.

¿Consideran que es posible?

Contesta mediante el chat de Zoom

Break

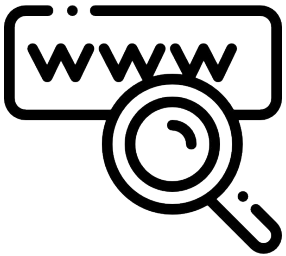
¡10 minutos y volvemos!

Mixin y Decoradores

Mixin

Mixin

Es posible que solo personas con login puedan ver nuestra página web, para ello existen los mixin y los decoradores que tienen como función validar lógica en nuestras views.



Si deseas que solo se pueda acceder a una Clase estando logueado podemos usar Mixin de Django:

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

class ClaseQueNecesitaLogin (LoginRequiredMixin):
```

💃 ¡Listo!, solo podrán hacer uso de la clase los usuarios registrados.

Decoradores

Decoradores

```python
#Decorador por defecto
from django.contrib.auth.decorators import login_required
```

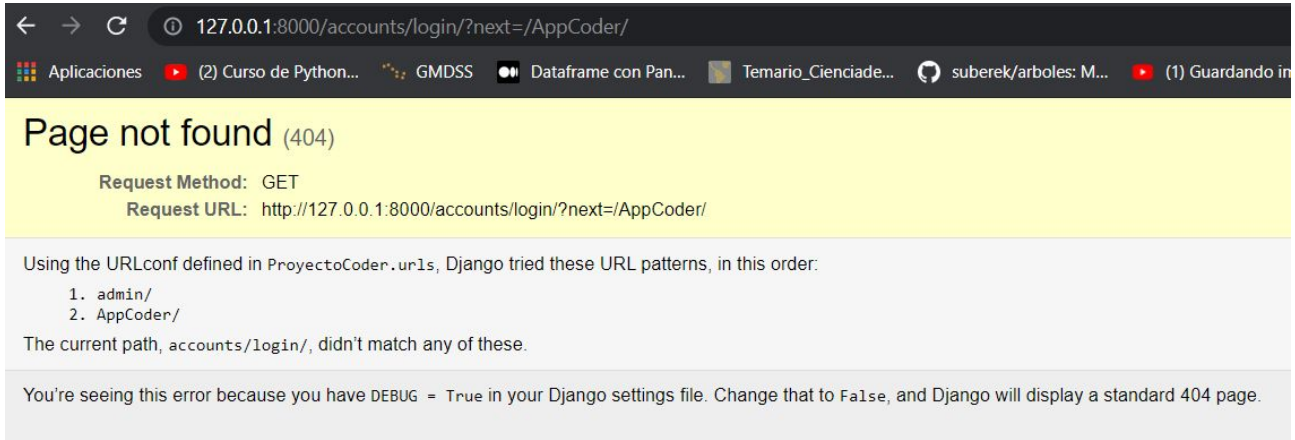
```python
@login_required
def inicio(request):

    return render(request, "AppCoder/inicio.html")
```

Nos sirven para validar identidad rápidamente, pero están más orientados a vistas y def.

👈 Primero usaremos el decorador más simple y ya dado por Django.

Decoradores

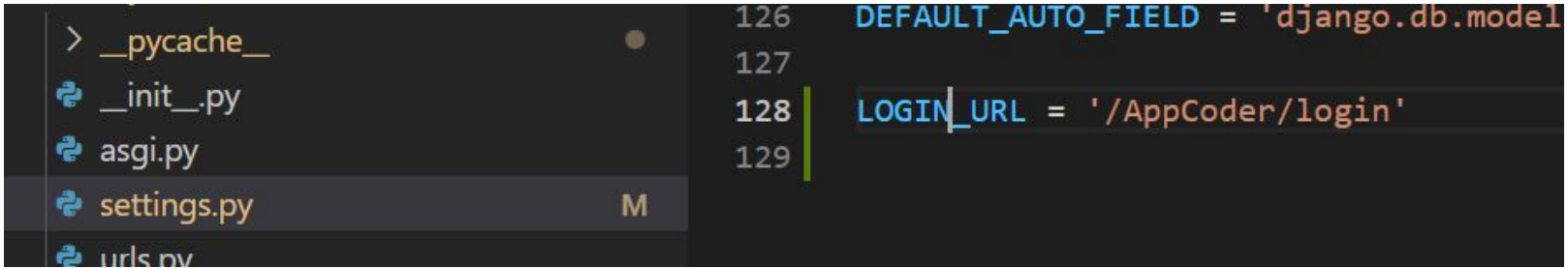
Al tratar entrar a la página de Inicio nos prohíbe el ingreso y nos sale lo siguiente 👇



CODERHOUSE

Decoradores

Eso es porque no estamos logueados, entonces nos quiere redireccionar a un lugar predefinido que no existe.
Lo tenemos que modificar desde settings.py para que nos lleve a nuestro login



¡LISTO! 💃

Ejemplos de decoradores y mixin

Aquí podrán encontrar más ejemplos de todo lo que se puede hacer con mixin y decorados.

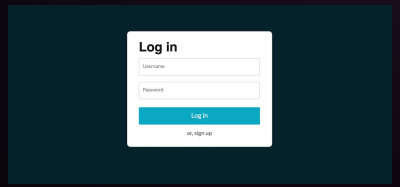
Para pensar

¿Qué más podríamos agregar al login?

Contesta mediante el chat de Zoom

Para pensar

La respuesta podemos verla en cualquier página relativamente compleja. Entren a su perfil de la plataforma de Coder, al perfil de Drive a su perfil de e-mail, etc. y eso sería generar un Login sofisticado, cosa que trataremos de hacer la próxima clase.



CODERHOUSE

Agregar login

Pensando en la entrega final, agrégale Login al proyecto donde la clase pasada hiciste CRUD.

Duración: **20 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Agregar login

Pensando en la entrega final, agrégale Login al proyecto donde la clase pasada hiciste CRUD. Principalmente, busca que se pueda crear el usuario y loguearse.

¿Preguntas?

Resumen
de la clase hoy

- ✓ Logueo, deslogueo y registro de nuestra web.
- ✓ Privacidad de la web mediante decoradores.

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación