

Clase 23. PYTHON

Playground Avanzado

Parte II

Esta clase va a ser

• grabad

a

Temario

22

Playground avanzado Parte I

- ✓ CRUD
- ✓ Clases basadas en vistas

23

Playground avanzado Parte II

- ✓ Login -
Registro -
Logout
- ✓ Mixin y
Decoradores

24

Playground Avanzado Parte III

- ✓ Edición de usuario
- ✓ Avatar
- ✓ Unit test

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](#)



**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.

Login

Login



editarProfesor.html
entregables.html
estudiantes.html
inicio.html M
leerProfesores.html
login.html U
padre.html
profesores.html
init.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py M
views.py M

```
9  <body>
10
11
12    <form action="" method="POST">
13
14      {% csrf_token %}
15
16      {{form.as_p}}
17
18      <button type="submit"> Iniciar Sesión</button>
19
20
21    </form>
22
23
24
25  </body>
26  </html>
```

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">Iniciar 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
```

Login



```
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](#)

Login



Resultados

127.0.0.1:8000/AppCoder/login

Aplicaciones (2) Curso de Python... GMDSS Datos

Username:

Password:

Iniciar Sesión

Datos correctos

Bienvenido nico_
Podes buscar una camada:

Datos incorrectos

Error, formulario erroneo
Podes buscar una camada:

Registro

Registro



```
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'),
```

The screenshot shows a code editor with a file tree on the left and a code editor window on the right. The file tree includes files like curso_confirm_delete.html, curso_detalle.html, curso_form.html, cursos_list.html, cursos.html, editarProfesor.html, entregables.html, estudiantes.html, inicio.html, leerProfesores.html, login.html, padre.html, profesores.html, and registro.html. The registro.html file is selected and shown in the code editor. The code in registro.html is:

```
7 |     <title>Registro</title>  
8 |     </head>  
9 |     <body>  
10 |  
11 |  
12 |     <form action="" method="POST">  
13 |  
14 |         {% csrf_token %}  
15 |  
16 |         {{form.as_p}}  
17 |  
18 |         <button type="submit"> Registrarse!!!!</button>  
19 |  
20 |  
21 |     </form>  
22 |  
23 |  
24 |  
25 |     </body>  
26 |     </html>
```

Modificamos las **urls** y creamos
el **template**



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 😎

← → ⌛ ⓘ 127.0.0.1:8000/AppCoder/register

Aplicaciones (2) Curso de Python... GMDSS Dataframe con Pan... Temario_Cienciade... suberek/

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

[Iniciar Sesión](#)

Usuario Creado :)

Podes buscar una camada:



Registro

Funcionó, pero por defecto no se ve muy estético 😞

← → C ① 127.0.0.1:8000/AppCoder/register

Aplicaciones (2) Curso de Python... GMDSS Datafra

Username:

Email:

Contraseña:

Repetir la contraseña:

Lo arreglaremos para que se vea

así





Registro

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

```
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

```
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'),
```



Logout

OPEN EDITORS

PROYECTOCODER

- AppCoder
- _pycache_
- migrations
- static
- templates \ AppCoder
 - curso_confirm_delete.html
 - curso_detalle.html
 - curso_form.html
 - cursos_list.html
 - cursos.html
 - editarProfesor.html
 - entregables.html
 - estudiantes.html
 - inicio.html
 - leerProfesores.html
 - login.html
 - logout.html**
 - padre.html
 - profesores.html
 - registro.html
- init .py

Creamos el logout.html 😊

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

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

```
@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



The screenshot shows a browser window with the following details:

- Address bar: 127.0.0.1:8000/accounts/login/?next=/AppCoder/
- Tab bar: Aplicaciones, (2) Curso de Python..., GMDSS, Dataframe con Pan..., Temario_Cienciade..., suberek/arboles: M..., (1) Guardando in...
- Error message: Page not found (404)
- Request Method: GET
- Request URL: http://127.0.0.1:8000/accounts/login/?next=/AppCoder/
- Text below message:

Using the URLconf defined in `ProyectoCoder.urls`, Django tried these URL patterns, in this order:

 1. `admin/`
 2. `AppCoder/`

The current path, `accounts/login/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.



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

The screenshot shows a code editor with a dark theme. On the left, there is a sidebar with file icons: __pycache__, __init__.py, asgi.py, settings.py (which is highlighted in yellow), and urls.py. The main area shows the contents of settings.py:

```
126 DEFAULT_AUTO_FIELD = 'django.db.models.  
127  
128 LOGIN_URL = '/AppCoder/login'  
129
```

¡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.





Agregar login

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

Duración: **20 minutos**



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