Formularios Genéricos
Formularios Modelos

¿Qué tipo de Formulario uso?
Usaremos el Model Form, para crear los formularios de los modelos de forma más rápida

Usaremos los Formularios Genéricos, cuando tengamos que usar diferentes modelos en el formulario, o si necesitamos personalizar un modelo más concreto.

para el CRUD usaremos los formularios de Modelos

Create
Crear modelo

Read
Listar y ver Modelo
Filtro de búsqueda Avanzada → Formulario Genérico

Update
Actualizar Modelo

Delete
Eliminar Modelo

```
git branch CRUD_1
git checkout CRUD_1
```

CRUD

El la carpeta de nuestro proyecto creamos el archivo **forms.py**

```
from django import forms
from django.forms import ModelForm
from .models import *
```

from django import forms →Formulario Generico
from django.forms import ModelForm →Formulario Modelos

– crear formulario →**forms.py**

```
class UsuarioForm(ModelForm):
    class Meta:
        model = Usuario
        fields = ["nombre_usuario","correo","password"]
        labels = {
            "nombre_usuario":("Nombre de Usuario"),
            "correo":("Correo electrónico"),
            "password":("Contraseña"),
        }
        help_texts = {
            "nombre_usuario":("50 caracteres como máximo"),
        }
```

Una vez creado el formulario sólo tenemos que crear una Vista que lo envíe a una Template y mostrarlo en la Template: **views.py**

```
from .forms import *

#-------- USUARIO --------
def usuario_create(request):
    formulario = UsuarioForm()
    return render(request, 'usuarios/crud/create_usuario.html',{'formulario':formulario})
```

---

urls.py

```
    #---Usuario---
    path('usuario/crear',views.usuario_create, name='usuario_create'),
```

---

añade la url al index.html como hemos hecho con las demás url del proyecto

```
<li><a href="{% url 'usuario_create' %}">Crear USUARIO</a></li>
```

create_usuario.html →Sin usar Bootstrap

```
{% extends "base/base.html" %}
{% load static %}

{% block title %}ConcurNet - Crear Usuario{% endblock %}


{% block cabecera %}
   <h1>Crear Usuario</h1>
   <hr>
   <br>
{% endblock %}
{% block contenido %}
   <form action="" method="post">
       {% csrf_token %}
       {{formulario}}
       <button type="submit">Enviar</button>
   </form>
{% endblock %}
```

Esto nos da el formulario sin estilo, para ello instalaremos la libreria de Bootstrap

---

requirements.txt

```
django-bootstrap5~=24.3
django-bootstrap-icons~=0.8.6
```

pip install -r requirements.txt

mysite / settings.py → `INSTALLED_APPS`

```
   'django_bootstrap5',
   'django_bootstrap_icons',
```

Para usar la librería en las Templates debemos añadir:
{% load django_bootstrap5 %}
Para usar las clases de Bootstrap, debemos incluir su CSS:
{% bootstrap_css %}
Para usar el javascript de Bootstrap, debemos incluir su JS:
{% bootstrap_javascript %}
Para usar los iconos de bootstrap, debemos incluir:
{% load bootstrap_icons %}

Los `{% load %}` siempre van al inicio de tu template, antes de cualquier HTML.
Después, los tags como `{% bootstrap_css %}` y `{% bootstrap_javascript %}` se suelen poner dentro del `<head>` y al final del `<body>` respectivamente.

base.html →Archivo principal donde heredan todas las templates

```
{% load static %}
{% load django_bootstrap5 %}
{% load bootstrap_icons %}

<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <title>{% block title %}ConcurNet{% endblock %}</title>
   <link rel="stylesheet" href="{% static 'css/style.css' %}">
   <link rel="icon" type="image/png" href="{% static 'img/ConcurNet-7.png' %}">
   <script src="{% static 'js/app.js' %}" defer></script>

   {% bootstrap_css %}
    {% bootstrap_javascript %}


</head>
<body>

   {% include "base/_header.html" %}

   <main class="container">
       {% block cabecera %}
       {% endblock %}

       {% block contenido %}
       {% endblock %}
   </main>

   {% include "base/_footer.html" %}

</body>
</html>
```

create_usuario.html →Usando Bootstrap

```
{% extends "base/base.html" %}
{% load static %}
{% load django_bootstrap5 %}
{% load bootstrap_icons %}

{% block title %}ConcurNet - Crear Usuario{% endblock %}

{% block cabecera %}
   <h1>Crear Usuario</h1>
   <hr>
   <br>
{% endblock %}
{% block contenido %}
   <form action="" method="post">
       {% csrf_token %}
       {% bootstrap_form formulario %}
       <button type="submit">Enviar</button>
   </form>
{% endblock %}
```

Ahora volvemos a **views.py**
Completamos la vista que creamos antes para que pueda crear los modelos en la base de datos

```
from django.contrib import messages
from django.shortcuts import redirect
```

```
#-------- USUARIO --------
def usuario_create(request): # Metodo que controla el Tipo de formulario

    # Si la petición es GET se creará el formulario Vacío
    # Si la petición es POST se creará el formulario con Datos.
    datosFormulario = None
    if request.method == "POST":
        datosFormulario = request.POST

    formulario = UsuarioForm(datosFormulario)

    if (request.method == "POST"):

        usuario_creado = crear_usuario_modelo(formulario)

        if(usuario_creado):
            messages.success(request, 'Se ha creado el Usuario: [ '+formulario.cleaned_data.get('nombre_usuario')+" ] correctamente.")
            return redirect('home')

    return render(request, 'usuarios/crud/create_usuario.html',{'formulario':formulario})

def crear_usuario_modelo(formulario): # Metodo que crea en la base de datos

    usuario_creado = False
    # Comprueba si el formulario es válido
    if formulario.is_valid():
        try:
            # Guarda el usuario en la base de datos
            formulario.save()
            usuario_creado = True
        except Exception as error:
            print(error)
    return usuario_creado

#--------------------------
```

Ahora crearemos las Validaciones, para ello volvemos a el archivo **forms.py**
Y ampliamos nuestro formulario

```
class UsuarioForm(ModelForm):

   class Meta:
       model = Usuario
       fields = ["nombre_usuario","correo","password"]
       labels = {
           "nombre_usuario":("Nombre de Usuario"),
           "correo":("Correo electrónico"),
           "password":("Contraseña"),
       }
       help_texts = {
           "nombre_usuario":("50 caracteres como máximo"),
       }

    def clean(self):

        #Validamos con el modelo actual
        super().clean()

        #Obtenemos los campos
        nombre_usuario = self.cleaned_data.get('nombre_usuario')
        password = self.cleaned_data.get('password')

        #Comprobamos
        if len(nombre_usuario) < 3:
            self.add_error('nombre_usuario','El nombre debe tener al menos 3 caracteres.')

        if len(password) < 8:
            self.add_error('password','La contraseña debe tener al menos 8 caracteres.')

        #Siempre devolvemos el conjunto de datos.
        return self.cleaned_data
```

Por ultimo volvemos a **base.html** →Archivo principal donde heredan todas las templates
y añadimos la estructura que controla los mensajes como:

views.py

```
if(usuario_creado):
    messages.success(request, 'Se ha creado el Usuario: [ '+formulario.cleaned_data.get('nombre_usuario')+" ] correctamente.")
```

base.html →Archivo principal donde heredan todas las templates

```
<body>

  {% include "base/_header.html" %}

  {% bootstrap_messages %}

  <main class="main">
      {% block cabecera %}
      {% endblock %}

      {% block contenido %}
      {% endblock %}
  </main>

  {% include "base/_footer.html" %}

</body>
```

CRUD

Ahora crearemos un filtro de búsqueda SIMPLE

Este paso no se pide en la tarea

pero hazlo al menos en un modelo para después entender mejor el filtro AVANZADO

---

urls.py

```
  #--------------CRUD--------------
  #---Usuario---
  path('usuario/crear/',views.usuario_create, name='usuario_create'),
  path('usuario/buscar/',views.usuario_buscar, name='usuario_buscar'),
```

---

views.py

```
def usuario_buscar(request): # Busqueda Simple

  texto = request.GET.get("textoBusqueda", "")  # vacio si no se envia nada

  # trim
  texto = texto.strip()

  if texto:
      usuarios = Usuario.objects.filter(
          Q(nombre_usuario__icontains=texto) |
          Q(correo__icontains=texto)
      )
  else:
      usuarios = Usuario.objects.all()

  return render(request,'usuarios/lista_usuarios.html',{'Usuarios_Mostrar': usuarios,'Texto_Busqueda': texto,})
```

lista_usuarios.html

```
{% extends "base/base.html" %}
{% load static %}
{% load django_bootstrap5 %}
{% load bootstrap_icons %}

{% block title %}ConcurNet - Usuarios{% endblock %}

{% block cabecera %}
   <h1>Listado de Usuarios</h1>
{% endblock %}

{% block contenido %}

   <form action="{% url 'usuario_buscar' %}" method="GET" class="d-flex mt-3">
       <input
           name="textoBusqueda"
           class="form-control me-2"
           type="search"
           placeholder="Buscar usuario..."
           value="{{ Texto_Busqueda }}"
       >
       <button class="btn btn-light btn-outline-dark" type="submit">Buscar</button>
   </form>

   {% if Texto_Busqueda %}
       <p>Usuarios con "{{Texto_Busqueda}}" en su nombre o correo</p>
   {% endif %}

   <hr>

   <div class="list">
       {% for Usuario_Mostrar in Usuarios_Mostrar %}
           {% include "usuarios/_item_usuario.html" %}
       {% empty %}
           <h2>No se encontraron usuarios</h2>
       {% endfor %}
   </div>

{% endblock %}
```

Ahora crearemos un **Filtro de búsqueda AVANZADO**

Nos vamos al archivo de **forms.py** y crearemos un formulario genérico con todos los campos que queremos usar para filtrar

La tarea pide:
Leer(Busqueda Avanzada) (0.5) (***Usar tres campos en la búsqueda***)

En mi caso he usado **5**

forms.py

```
class UsuarioBuscarAvanzada(forms.Form):

    nombre_usuario_contiene = forms.CharField(
        label='Nombre de usuario contiene',
        help_text="(Opcional)",
        required=False
    )

    correo_contiene = forms.CharField(
        label='Correo contiene',
        help_text="(Opcional)",
        required=False
    )

    fecha_registro_desde = forms.DateField(
        label='Registro desde',
        help_text="(Opcional)",
        required=False,
        widget=forms.DateInput(attrs={'type':'date'})
    )

    fecha_registro_hasta = forms.DateField(
        label='Registro hasta',
        help_text="(Opcional)",
        required=False,
        widget=forms.DateInput(attrs={'type':'date'})
    )

    tipo_usuario = forms.ChoiceField(
        label='Tipo de usuario',
        help_text="Seleccione un tipo o 'Todos'",
        choices=[
            ('todos', 'Todos'),
            ('admin', 'Administrador'),
            ('jurado', 'Jurado'),
            ('participante', 'Participante'),
        ],
        required=True,
        initial='todos'
    )
```

Justo debajo añadimos el método `def clean(self):` para realizar las validaciones de este formulario.

```python
def clean(self):

        #Validamos con el modelo actual
        super().clean()

        #Obtenemos los campos
        nombre_usuario_contiene = self.cleaned_data.get('nombre_usuario_contiene')
        correo_contiene = self.cleaned_data.get('correo_contiene')
        fecha_registro_desde = self.cleaned_data.get('fecha_registro_desde')
        fecha_registro_hasta = self.cleaned_data.get('fecha_registro_hasta')
        tipo_usuario = self.cleaned_data.get('tipo_usuario')

        #Comprobamos
        if(tipo_usuario == ""):
            self.add_error('tipo_usuario','Seleccione un tipo de Usuario.')

        if(
            not fecha_registro_desde is None and
            not fecha_registro_hasta is None and
            fecha_registro_hasta < fecha_registro_desde
            ):
            self.add_error('fecha_registro_desde','Rango de fecha no valido.')
            self.add_error('fecha_registro_hasta','Rango de fecha no valido.')

        #Siempre devolvemos el conjunto de datos.
        return self.cleaned_data
```

IMPORTANTE en los campos que quieres que sean obligatorios:
Los que tienen:

```python
required=True,
```

Debes realizar una validación para comprobar si ese campo está vacío y si lo está, dar un error:

```python
if(tipo_usuario == ""):
            self.add_error('tipo_usuario','Seleccione un tipo de Usuario.')
```

Ejemplo de jorge

```python
#Controlamos los campos
if(textoBusqueda == ""
   and len(idiomas) == 0
   and fecha_desde is None
   and fecha_hasta is None
```

Yo no valido los campos opcionales porque después en **views.py** compruebo si están vacios, y si lo estan simplemente no aplico ningún filtro

---

Creamos la url

urls.py

```python
path('usuario/buscar/avanzado/',views.usuario_buscar_avanzado, name='usuario_buscar_avanzado'),
```

views.py

```
def usuario_buscar_avanzado(request): #Busqueda Avanzada

   if(len(request.GET) > 0):
       formulario = UsuarioBuscarAvanzada(request.GET)
       if formulario.is_valid():
           #Aqui vamos a ir aplicando filtros a la QS poco a poco
   else:
       formulario = UsuarioBuscarAvanzada(None)
   return render(request, 'usuarios/crud/buscar_avanzada_usuarios.html',{'formulario':formulario})
```

Tendremos la QS que es la consulta a la base de datos, que se construira poco a poco
el Mensaje de búsqueda, que indica que filtros hemos aplicado

Y todos los datos del formulario en variables que podremos comprobar

(todo el código que hay abajo va dentro de este if)

```
if formulario.is_valid():

           mensaje_busqueda = 'Filtros Aplicados:\n'
           QsUsuarios = Usuario.objects

           #Obtenemos los campos del formulario
           nombre_usuario_contiene = formulario.cleaned_data.get('nombre_usuario_contiene')
           correo_contiene = formulario.cleaned_data.get('correo_contiene')
           fecha_registro_desde = formulario.cleaned_data.get('fecha_registro_desde')
           fecha_registro_hasta = formulario.cleaned_data.get('fecha_registro_hasta')
           tipo_usuario = formulario.cleaned_data.get('tipo_usuario')
```

Aquí hago un if que si el campo del formulario **nombre** no está vacío (tiene algo escrito), que filtre a los **usuarios** por el atributo **nombre_usuario** con un __icontains
si no (else), No hace ningún filtro y lo indico en el mensaje

el **.strip()** es equivalente al TRIM (Elimina espacios del principio y final)

```
           #---Nombre---
          if(nombre_usuario_contiene!=''):
              nombre_usuario_contiene = nombre_usuario_contiene.strip()
              QsUsuarios = QsUsuarios.filter(nombre_usuario__icontains=nombre_usuario_contiene)
              mensaje_busqueda += '· Nombre contiene "'+nombre_usuario_contiene+'"\n'
           else:
              mensaje_busqueda += '· Cualquier nombre \n'
```

Aquí hago un if que si el campo del formulario **correo** no está vacío (tiene algo escrito), que filtre a los **usuarios** por el atributo **correo** con un __icontains
si no (else), No hace ningún filtro y lo indico en el mensaje

el **.strip()** es equivalente al TRIM (Elimina espacios del principio y final)

```
        #---Correo---
    if(correo_contiene!=''):
            correo_contiene = correo_contiene.strip()
            QsUsuarios = QsUsuarios.filter(correo__icontains=correo_contiene)
            mensaje_busqueda += '· Correo contiene "'+correo_contiene+'"\n'
    else:
            mensaje_busqueda += '· Cualquier correo \n'
```

Aquí hago un if que si el campo del formulario **fecha_registro_desde** no es None (tiene una fecha), que filtre a los **usuarios** por el atributo **fecha_registro** con un __gte

si no (else), No hace ningún filtro y lo indico en el mensaje

```
        #---Fecha---
    if (not fecha_registro_desde is None):
        QsUsuarios = QsUsuarios.filter(fecha_registro__gte=fecha_registro_desde)
        mensaje_busqueda += '· Registro desde '+datetime.strftime(fecha_registro_desde,'%d-%m-%Y')+'\n'
    else:
        mensaje_busqueda += '· Registro desde: Cualquier fecha \n'
```

Aquí hago un if que si el campo del formulario **fecha_registro_hasta** no es None (tiene una fecha), que filtre a los **usuarios** por el atributo **fecha_registro** con un __lte

si no (else), No hace ningún filtro y lo indico en el mensaje

```
    if (not fecha_registro_hasta is None):
        QsUsuarios = QsUsuarios.filter(fecha_registro__lte=fecha_registro_hasta)
        mensaje_busqueda += '· Registro hasta '+datetime.strftime(fecha_registro_hasta,'%d-%m-%Y')+'\n'
    else:
        mensaje_busqueda += '· Registro hasta: Cualquier fecha \n'
```

Tipos de Operadores en el filter **PDF** Página 4

Aquí hago if del campo del formulario **tipo_usuario.**
Si el valor corresponde a un rol concreto (administrador, jurado o participante), aplicó un filtro que busca usuarios cuyo modelo relacionado 1:1 no sea nulo. (que tenga una FK != null)
Si se elige "todos", no aplico ningún filtro y lo indico en el mensaje.

```
            #---Tipo---
            if tipo_usuario == 'admin':
                QsUsuarios = QsUsuarios.filter(administrador__isnull=False)
                mensaje_busqueda += '· Rol: Admin \n'

            elif tipo_usuario == 'jurado':
                QsUsuarios = QsUsuarios.filter(jurado__isnull=False)
                mensaje_busqueda += '· Rol: Jurado \n'

            elif tipo_usuario == 'participante':
                QsUsuarios = QsUsuarios.filter(participante__isnull=False)
                mensaje_busqueda += '· Rol: Participante \n'

            else:
                mensaje_busqueda += '· Rol: Cualquiera \n'
```

Por último terminamos la QS con un .all() porque es una lista de usuarios y retornamos al HTML la lista de usuarios con todos los filtros aplicados anteriormente, y el mensaje de busqueda

```
            #Ejecutamos la querySet y enviamos los usuarios
            usuarios = QsUsuarios.all()

            return render(request, 'usuarios/lista_usuarios.html',
                        {'Usuarios_Mostrar':usuarios,
                        'Mensaje_Busqueda':mensaje_busqueda}
                        )
```

---

recuerda que todo el código que hemos estado viendo va dentro del `if formulario.is_valid():`

```
if(len(request.GET) > 0):
      formulario = UsuarioBuscarAvanzada(request.GET)
      if formulario.is_valid():
            #Aqui vamos a ir aplicando filtros a la QS poco a poco
else:
```

---

lista_usuarios.html

La etiqueta <pre></pre> es para que los \n se apliquen si le pones es lo mismo pero sin saltos de linea

```
{% extends "base/base.html" %}
{% load static %}

{% block title %}ConcurNet - Usuarios{% endblock %}

{% block cabecera %}
   <h1>Listado de Usuarios</h1>
{% endblock %}

{% block contenido %}

   {% if Mensaje_Busqueda %}
        <pre>{{Mensaje_Busqueda}}</pre>
    {% endif %}

   <div class="list">
       {% for Usuario_Mostrar in Usuarios_Mostrar %}
           {% include "usuarios/_item_usuario.html" %}
       {% empty %}
           <h2>No se encontraron usuarios</h2>
       {% endfor %}
   </div>
{% endblock %}
```

CRUD

urls.py

```
path('usuario/editar/<int:id_usuario>', views.usuario_editar, name="usuario_editar"),
```

views.py

```
def usuario_editar(request, id_usuario):
   usuario = Usuario.objects.get(id = id_usuario)

   # Si la petición es GET se creará el formulario Vacío
   # Si la petición es POST se creará el formulario con Datos.
   datosFormulario = None
   if request.method == "POST":
       datosFormulario = request.POST

   formulario = UsuarioForm(datosFormulario,instance=usuario)

   if (request.method == "POST"):

       usuario_creado = crear_usuario_modelo(formulario)

       if(usuario_creado):
       messages.success(request, 'Se ha actualizado el Usuario: [ '+formulario.cleaned_data.get('nombre_usuario')+" ] correctamente.")
           return redirect('usuario_buscar')

  return render(request, 'usuarios/crud/actualizar_usuario.html', {'formulario':formulario,'usuario':usuario})
```

_item_usuario.html

CRU**D**

```
path('usuario/eliminar/<int:id_usuario>', views.usuario_eliminar, name="usuario_eliminar"),
```