TEMA IV

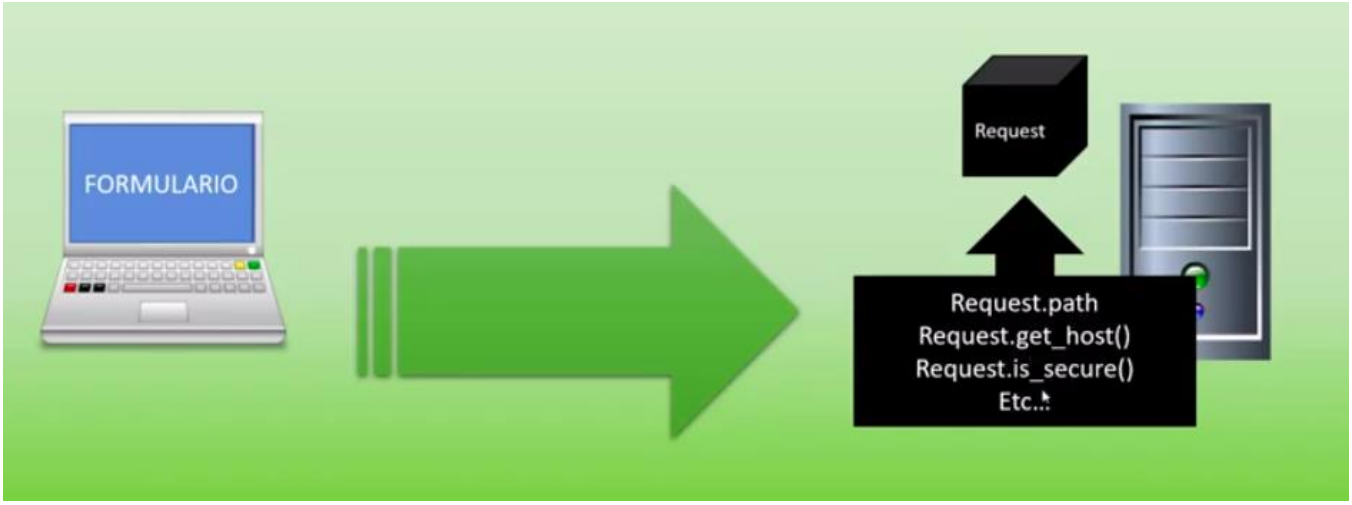
DESARROLLO DE APLICACIONES WEB EN BACKEND

"Siempre parece imposible,
hasta que se hace"
Nelson Mandela

FUNDAMENTOS DE LA PROGRAMACIÓN WEB

7.6 FORMULARIOS

Los formularios son elementos fundamentales en las aplicaciones web, son la herramienta que se usa para enviar datos desde el cliente al servidor:



Los principales pasos que hace el proceso del manejo de formularios de Django son:

1. Mostrar el formulario predeterminado la primera vez que es solicitado por el usuario.

- El formulario puede contener campos en blanco (por ejemplo, si está creando un registro nuevo), o puede estar rellenado previamente con valores iniciales (por ejemplo, si está modificando un registro o si tiene valores iniciales predeterminados útiles).
- El formulario se conoce como no vinculado en este punto porque no está asociado con ningún dato ingresado por el usuario (aunque pueda tener valores iniciales).

2. Recibir datos de una solicitud de envío y vincularlo al formulario.

- La vinculación de datos al formulario significa que los datos ingresados por el usuario y cualquier erro están disponibles cuando necesitamos volver a desplegar el formulario.

3. Limpiar y validar los datos. Clean and validate the data.

- La limpieza de los datos, por ejemplo, borrar caracteres no válidos que podrían ser usados para enviar contenido malicioso al servidor y los convierte en tipos consistente de Python.
- La validación verifica que los valores sean apropiados para el campo (por ejemplo, que estén en el rango correcto de fechas, no sean demasiado cortos ni demasiado largos, etc.)

4. Si algún dato es no válido, volver a mostrar el formulario, esta vez con cualquier valor rellenado por el usuario y los mensajes de error para los campos con problemas.

5. Si todos los datos son válidos, realizar las acciones requeridas (por ejemplo, guardar los datos, enviar un correo electrónico, devolver el resultado de una búsqueda, cargar un archivo, etc)

6. Una vez todas las acciones se hayan completado, redirigir al usuario a otra página

Django provee una serie de herramientas para ayudar con estas tareas. La más fundamental es la clase Form, que simplifica la generación de formularios HTML, la limpieza y validación de datos.

Veamos primero como enviar y recibir datos generando nosotros el html necesario.

Recogiendo la información de los controles HTML

Cuando un navegador cliente envía una petición al servidor, que contiene la información introducida por el usuario en un formulario, al servidor dicha petición le llega mediante un objeto de tipo Request.

Dicho objeto almacena, además, información sobre el cliente: ip, navegador, si la conexión es segura….

En la página oficial de Django podemos consultar todos los atributos y métodos de la clase HttRequest, veamos algunos de ellos.

El objeto request tienen dos atributos, definidos como diccionarios, GET y POST que se utilizan para almacenar la información que nos llega desde el cliente a través de los controles HTML del formulario.

Cada vez que se envía una petición se genera un diccionario cuyas claves son los nombres de los controles html, siendo el valor de dichos controles, el valor que se almacena asociado a la clave.

El objeto request tiene un método que nos permiten saber cuál de estos envíos ha utilizado el cliente.

HttpRequest.method

En HTML, independientemente del lenguaje de programación o del método usado para hacer la petición, los datos desde un navegador nos llegan de la siguiente forma:

1. Los campos de tipo input=text nos llegan como una variable string cuyo nombre es el atributo name del control y su contenido lo que haya escrito el usuario

2. Para un control select, recibiremos una variable cuyo nombre es el atributo name del control y el valor el atributo value del elemento seleccionado.

3. Para radiobutton, recibiremos una variable cuyo nombre es el atributo name y el valor el atributo value del seleccionado

4. Para checkbox, recibiremos una lista cuyo nombre es el atributo name de los checkbox (todos deben llamarse igual). En la lista se incluyen todos los atributos value de los seleccionados.

Ejemplo: Página en la que mostramos una caja con nuestro nombre y edad y mostramos en otra un mensaje de bienvenida según la edad que tengamos.

```
def home(request):
    return render(request,"home.html")
    #return HttpResponse(loader.get_template('home.html').render())


def saludo(request):
    nombre=request.GET['nombre']
    edad=int(request.GET['edad'])
    if (edad<18):
        mensaje=", Lo sentimos. No puedes acceder."
    else:
        mensaje=", Bienvenida"
    return render(request,'saludo.html',{"nombre":nombre,"msg":mensaje})

"""
```

Podríamos tener una única vista que controle si es la primera vez que cargamos el formulario o las siguientes.

Un formulario puede "autollamarse", de manera que la primera vez se carga la plantilla sin más y las siguientes veces tenemos que detectar que nos han "envidado" el objeto request con los datos introducidos por el cliente

Ejemplo: Repetimos el ejemplo anterior con una sola vista

```
def home(request):
    if request.GET:
        nombre=request.GET['nombre']
        edad=int(request.GET['edad'])
        if (edad<18):
            mensaje=", Lo sentimos. No puedes acceder."
        else:
            mensaje=", Bienvenida"
        return render(request,'saludo.html',{"nombre":nombre,"msg":mensaje})
    else:
        return render(request,"home.html")
```

Análogamente podemos usar POST

Ejemplo: Cargamos una página de contacto, la misma vista la carga inicialmente y después nos da las gracias.

```
@csrf_exempt
def homePOST(request):
    if request.method=="POST":
        nombre=request.POST['nombre']
        email=request.POST['email']
        texto=request.POST['texto']
        return render(request,'saludo.html',locals())
    else:
        return render(request,"homepost.html")
```

Para saber qué es CSRF consulta la siguiente web

https://www.stackhawk.com/blog/django-csrf-protection-guide/

Aunque se puede trabajar con formularios html, Django incorpora un api para generarlos, incluye métodos que nos facilitan las comprobaciones y validaciones.

FORM API

Este api nos proporciona una sencilla forma de generar formularios y además métodos que nos facilitan la validación y el saneamiento de los datos recibidos.

Podemos consultar la documentación de django para ver todas las posibilidades que nos ofrece.

Para trabajar con este api:

1. Creamos un archivo llamado forms.py, dentro de él definimos una clase por cada formulario que queremos generar. Se puede crear en cualquier punto de nuestro proyecto, pero por convención se suele poner en el mismo sitio en el que tengamos views.py.

2. Importamos forms de django

3. Definimos la clase, con el nombre que queramos, hereda de forms.Form

Definimos un atributo de clase para cada control HTML. Por defecto crea los controles con la propiedad required a True, sólo la cambiaremos si no queremos que el campo sea obligatorio. También añade el atributo id de forma automática

```python
from django import forms

class FormularioContacto(forms.Form):
    nombre=forms.CharField(max_length=25,required=True)
    email=forms.EmailField(max_length=25,required=True)
    mensaje=forms.CharField(max_length=100,required=True)
```

Además, en esas variables, guardaremos la información que nos llega desde el cliente.

Para pasar datos al formulario desde la vista, podemos pasar un diccionario con esos campos y sus valores

4. En la correspondiente vista instanciamos una variable de este tipo para que django genere el formulario.

Tenemos que validar todos los datos que nos llegan, para ello usaremos el método is_valid().

Para acceder a los datos que nos llegan, una vez comprobado que son válidos tenemos la propiedad cleaned_data que nos retorna un diccionario tal y como hace el objeto request con POST o GET

```python
from django.shortcuts import render
from formulario.forms import FormularioContacto

# Create your views here.
def home(request):
    if request.POST:
        miFrm=FormularioContacto(request.POST)
        if miFrm.is_valid():
            dicc=miFrm.cleaned_data
            return render(request, 'datos.html',dicc)



    else:
        miFrm=FormularioContacto()

    return render(request, "home.html",{"form":miFrm})
```

5. Sólo nos queda hacer las plantillas

```
<body>
    <div class="contenedor">
        <form action="" method="POST">
            {{form}}
            <input type="submit" value="Aceptar" class="btn bg-dark">
        </form>

    </div>
</body>
```

Si lo probamos podemos comprobar que genera los controles html asociados a las variables de la clase. No genera las etiquetas form ni el submit. Tampoco aplica ningún formato.

Pero las validaciones las hace de forma automática.

Tipos de campos

Hay muchos otros tipos de campos de formulario, que reconocerá en gran medida por su similitud con las clases de campo de modelo equivalentes: `BooleanField`, `CharField`, `ChoiceField`, `TypedChoiceField`, `DateField`, `DateTimeField`, `DecimalField`, `DurationField`, `EmailField`, `FileField`, `FilePathField`, `FloatField`, `ImageField`, `IntegerField`, `GenericIPAddressField`, `MultipleChoiceField`, `TypedMultipleChoiceField`, `NullBooleanField`, `RegexField`, `SlugField`, `TimeField`, `URLField`, `UUIDField`, `ComboField`, `MultiValueField`, `SplitDateTimeField`, `ModelMultipleChoiceField`, `ModelChoiceField`.

Los que con más frecuencia usaremos:

- CharField: campos de tipo texto (text, password). Usando los widgets podemos determinar si es texto, password y otros atributos.

- EmailField: lo que su nombre indica. Valida que la dirección tenga el formato correcto.
- IntegerField,DeciamlField,FloatField: para números porque valida formato. En el caso de los decimales podemos decirle cuantos queremos.
- ChoiceField:para controles tipo select, radiobuttons o un sólo checkbox. Es para controles de selección única. Con los widgets le diremos de que tipo queremos el control
- MultipleChoiceField: para controles de selección múltiple, varios checkbox
- DateField, DateTimeField: para controles de fecha y hora.

Los argumentos comunes a la mayoría de estos tipos son:

required: Si es True, el campo no se puede dejar en blanco o dar un valor None. Los Campos son obligatorios por defecto.

label: label es usado cuando renderizamos el campo en HTML. Si no se especifica entonces Django crearía uno a partir del nombre del campo al poner en mayúscula la primera letra y reemplazar los guiones bajos por espacios

label_suffix: Por defecto, se muestran dos puntos después de la etiqueta. Este argumento le permite especificar un sufijo que contiene otros caracteres.

initial: El valor inicial para el campo cuando es mostrado en el formulario.

Widget: El widget de visualización para usar.

help_text: texto adicional que se puede mostrar en formularios para explicar cómo usar el campo.

error_messages: Una lista de mensajes de error para el campo. Puede reemplazarse con nuestros propios mensajes si es necesario.

validators: Una lista de funciones que se invocarán en el campo cuando se valide.

disabled: El campo se muestra, pero su valor no se puede editar si esto es True. Por defecto es False.

```python
class FormularioContacto(forms.Form):
    nombre=forms.CharField(widget=forms.TextInput(attrs= {'class':'form-control','placeholder':"Tu nombre"}),label="Nombre")
    email=forms.EmailField(widget=forms.EmailInput(attrs={'class':'form-control','placeholder':'ejemplo@ejemplo.com'}),label="E
    fecha=forms.DateField(widget=forms.SelectDateWidget(attrs={'class':'form-control'},years=range(1900,2002)))
    SEXO=[('mujer','Femenino'),
          ('hombre','Masculino')]
    sexo=forms.ChoiceField(choices=SEXO,widget=forms.RadioSelect)
    mensaje=forms.CharField(widget=forms.Textarea(attrs={'rows':6,'class':'form-control'}), initial="Dinos lo que piensas")
    autorizacion=forms.CharField(label="Autorizo el tratamiento de mis datos",widget=forms.CheckboxInput)
    #Este checkbox es para cuando ponemos uno sólo, también podemos usar ChoiceFiel. por defecto su valor es requerido por lo
    #que si no lo clikeamos no pasaremos la validación
    #Si queremos poner varios checkbox tenemos que usar MultipleChoiceField con el widget CheckboxSelectMultiple
    #Si queremos usar una lista(select) ChoiceField con widget select
```

Podemos recorrer la colección de campos de un formulario para añadirle algo de estilo al mismo.

Ejemplo: Añadiendo estilos al formulario del ejemplo anterior

```python
class FormularioContacto(forms.Form):
    nombre=forms.CharField(max_length=8,label="Nombre")
    email=forms.EmailField(label="Email")
    mensaje=forms.CharField(label="Tu opinión",widget=forms.Textarea(attrs={'rows':5,'cols':20}), initial="Dinos lo que piensas")
    forms.CharField()
```

```html
<div class="contenido">
    <h2 class="display-6">Deja tus datos</h2>
    <p class="lead">Nos pondremos en contacto contigo</p>
    <div class="jumbotron">
        <form action="" method="POST">
            {%for field in form%}
            <div class="form-group">
                <label for="{{field.name}}">{{field.label}}</label>
                <br> {{field}}
            </div>
            {%endfor%}
            <div class="form-group"></div>
            <input type="submit" class="btn btn-success btn-lg" value="Aceptar">
    </div>
    </form>
</div>
</div>
```

Widgets

Siempre que especificamos un campo en un formulario, Django utiliza un widget predeterminado y apropiado para el tipo de datos que se mostrarán. Podemos determinar a través de este atributo que widget queremos usar:

CharField(widget=forms.Textarea)

Muchos de estos widgets a su vez pueden tener atributos

CharField(widget=forms.Textarea(attrs={'rows':5,"cols":5})

También podemos utilizar attrs para asignarle una clase css

forms.CharField(widget=forms.TextInput(attrs={'class': 'special'}))

Podemos modificar la colección attrs

```
name = forms.CharField()
url = forms.URLField()
comment = forms.CharField()

name.widget.attrs.update({'class': 'special'})
comment.widget.attrs.update(size='40')
```

Algunos otros ejemplos de widgets:

```
BIRTH_YEAR_CHOICES = ['1980', '1981', '1982']
FAVORITE_COLORS_CHOICES = [
  ('blue', 'Blue'),
  ('green', 'Green'),
```

```
  ('black', 'Black'),
]


class SimpleForm(forms.Form):
  birth_year = forms.DateField(widget=forms.SelectDateWidget(years=BIRTH_YEAR_CHOICES))
  favorite_colors = forms.MultipleChoiceField(
    required=False,
    widget=forms.CheckboxSelectMultiple,
    choices=FAVORITE_COLORS_CHOICES,
```

Ejercicio: "Añadimos un checkbox al formulario anterior de contacto para indicar si autorizamos el tratamiento de nuestros datos". En la siguiente página mostramos además de los datos un mensaje indicando lo que hemos elegido.

```
class FormularioContacto(forms.Form):
    nombre=forms.CharField(max_length=8,label="Nombre")
    email=forms.EmailField(label="Email")
    autorizacion=forms.CharField(label="Autorizo el tratamiento de mis datos",required=False,widget=forms.CheckboxInput)
    mensaje=forms.CharField(label="Tu opinión",widget=forms.Textarea(attrs={'rows':5,'cols':20}), initial="Dinos lo que piensas")
    forms.CharField()
```

```
<body>
    <p>Nombre: {{nombre}}</p>
    <p>Email: {{email}}</p>
    <p>Tu mensaje: {{mensaje}}</p>
    <p>Autorización: {% ifequal autorizacion 'True' %} doy {%else%} no doy {%endifequal%} mi consentimiento
    </p>
</body>
```

A los widgets también le podemos pasar como atributo clases u atributos de html

Ejemplo: "Añadiremos el atributo placeholder, la clase form-control y algunos otros controles típicos"

```python
class FormularioContacto(forms.Form):
    nombre=forms.CharField(widget=forms.TextInput(attrs= {'class':'form-control','placeholder':"Tu nombre"}),label="Nombre")
    email=forms.EmailField(widget=forms.EmailInput(attrs={'class':'form-control','placeholder':'ejemplo@ejemplo.com'}),label="Email")
    fecha=forms.DateField(widget=forms.SelectDateWidget(attrs={'class':'form-control'},years=range(1900,2002)))
    SEXO=[('mujer','Femenino'),
          ('hombre','Masculino')]
    sexo=forms.ChoiceField(choices=SEXO,widget=forms.RadioSelect)
    mensaje=forms.CharField(widget=forms.Textarea(attrs={'rows':6,'class':'form-control'}), initial="Dinos lo que piensas")
    autorizacion=forms.CharField(label="Autorizo el tratamiento de mis datos",widget=forms.CheckboxInput)
```

Nuestra página html

```
    <style>
        .contenido {
            width: 40%;
            margin-left: auto;
            margin-right: auto;
            margin-top: 20px;
            padding: 5px;
        }

        li {
            display: inline-block;
            list-style: none;
            padding-right: 10px;
        }
    </style>
</head>

<body>

    <div class="contenido">
        <h2 class="display-6">Deja tus datos</h2>
        <p class="lead">Nos pondremos en contacto contigo</p>
        <div class="jumbotron">
            <form action="" method="POST">
                {%for field in form%}
                <div class="form-group">
                    {%ifnotequal field.name 'mensaje'%}
                    <label for="{{field.name}}">{{field.label}}</label> {%endifnotequal%} {{field}}
                </div>
                {%endfor%}
                <div class="form-group"></div>
                <input type="submit" class="btn btn-success btn-lg" value="Aceptar">
        </div>
        </form>
    </div>
    </div>
```

Además, mostraremos en la página datos.html todos lo que el usuario ha introducido. Utilizaremos plantillas para tener la base común.

En el ejemplo, hemos recorrido la colección de controles lo que nos obliga a preguntar por el textarea para no poner label. Ni siempre es buena idea recorrer la colección, a veces es conveniente poner los controles uno a uno.

Validación de los datos

Hay varias formas de insertar nuestras propias validaciones. Si vamos a usar nuestra regla una y otra vez, podemos crear un nuevo tipo de campo. Pero la mayor parte de las veces, sólo las usaremos una vez. Las vamos a añadir directamente a la clase del formulario.

Para ello añadiremos un método clean. Lo vemos con un ejemplo

Ejemplo: "Formulario para remitir incidencias"

Nuestra clase formulario tendrá un campo para el email, una lista de selección para el tipo de problema que tenemos y un textarea para dejar nuestra explicación

```
from django import forms

class ContactForm(forms.Form):
    email=forms.EmailField(widget=forms.TextInput(attrs={'placeholder':'noreply@example.com',
                                                         'class':'form-control'}))
    INCIDENCIAS=[('software', 'Software'),
                 ('hardware', 'Hardware' ),
                 ('red', 'Conexión a internet')]
    incidencia=forms.ChoiceField(choices=INCIDENCIAS,widget=forms.Select(attrs={'class':'form-control'}))
    mensaje=forms.CharField(widget=forms.Textarea(attrs={'class':'form-control'}))
```

Nuestra vista, cargará la primera vez el formulario vacío y cuándo recibe un objeto request mediante POST, procesa los datos y muestra otra página de confirmación de información recibida.

En este caso vamos a recoger los datos uno a uno y definiremos nosotros el diccionario. Esto es así porque no vamos a pasar a la otra página todos los datos.

En los ejemplos anteriores utilizábamos el método clean_data de manera que nos retorna el diccionario ya creado.

```python
def incidencia(request):
    if request.method=="POST":
        miFormulario=ContactForm(request.POST)
        if miFormulario.is_valid():
            email=miFormulario.cleaned_data['email']
            mensaje=miFormulario.cleaned_data['mensaje']
            dicc={'email':email,'mensaje':mensaje}
            return render(request,'datos.html',dicc)
    else:
        miFormulario=ContactForm()
    return render(request,'home.html',{'form':miFormulario})
```

Por último, diseñamos la plantilla, en este caso tampoco vamos a usar un bucle para "pintar" los controles. Al ser pocos, podemos hacerlo de manera individual.

```
{%extends 'base.html' %}
{%block titulo %} Indicanos tu problema {%endblock%}
{%block subtitulo%}Nos pondremos en contacto contigo{%endblock%}
{%block contenido %}
<form action="" method="POST">
    <div class="form-group">
        <label>Email *</label>
        <div class="input-group">{{form.email}}</div>
    </div>

    <div class="form-group">
        <label for="">Tipo Incidencia *</label>
        <div class="input-group">{{form.incidencia}}</div>
    </div>
    <div class="form-group">
        <label>Mensaje *</label>
        <div class="input-group">{{form.mensaje}}</div>
    </div>
    <br>
    <center>
        <input type="submit" class="btn btn-success btn-lg" value="Aceptar">
    </center>
    </div>
</form>
{%endblock%}
```

Ya sólo nos queda añadir nuestra regla de validación. Añadimos en la clase del formulario un método que compruebe nuestra regla: "El mensaje tiene que tener más de 5 palabras"

```
class ContactForm(forms.Form):
    email=forms.EmailField(widget=forms.TextInput(attrs={'placeholder':'noreply@example.com',
                                                         'class':'form-control'}))
    INCIDENCIAS=[('software', 'Software'),
                 ('hardware', 'Hardware' ),
                 ('red', 'Conexión a internet')]
    incidencia=forms.ChoiceField(choices=INCIDENCIAS,widget=forms.Select(attrs={'class':'form-control'}))
    mensaje=forms.CharField(widget=forms.Textarea(attrs={'class':'form-control'}))


    def clean_mensaje(self):
        mensaje=self.cleaned_data['mensaje']
        if len(mensaje.split())<5:
            raise forms.ValidationError("No son suficientes palabras.Por favor, especifique más")
        return mensaje
```

ValidationError es la clase que utiliza este api para generar las excepciones que se producen en la validación de los datos.

El método debe llamarse clean_propiedad y se le llama automáticamente desde el método is_valid de la clase form.

Si probamos el funcionamiento, veremos que no nos muestra error, pero si el mensaje no tiene la longitud suficiente no accedemos a la siguiente página. Esto quiere decir que nuestra validación funciona.

Cada atributo de nuestro formulario tiene la propiedad errors, es un diccionario:

- clave: el nombre del campo
- valor: una lista de cadenas con los mensajes de los errores

Ahí es donde se almacenan los errores asociados al mismo durante las validaciones. Por defecto, los mensajes de la validación del método is_valid se

muestran en nuestros formularios. Los nuestros no lo hacen, para que sea así añadiremos en nuestro html

```
{{form.mensaje.errors}}
```

Puesto que es una lista, si tenemos más de un mensaje podemos recorrerla para que se muestren.

El api genera una lista html (ul) para mostrarlos en la página. Si queremos aplicarles estilos debemos tener en cuenta esto.