Formularios

Jorge Bańon Abad

¿Qué es un formulario?

Un formulario web o formulario HTML, es un elemento de la página Web que permite a los usuarios ingresar información y enviarla a un servidor para su procesamiento

Formularios en HTML

Ejemplo de formulario HTML:

```
<form action = "{% url 'libro_create' %}" method = "post" enctype='multipart/form-data'>
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre_usuario" />
        <button type="submit">Envíe su mensaje</button>
</form>
```

Formularios en HTML

Entendamos cada parte del formulario:

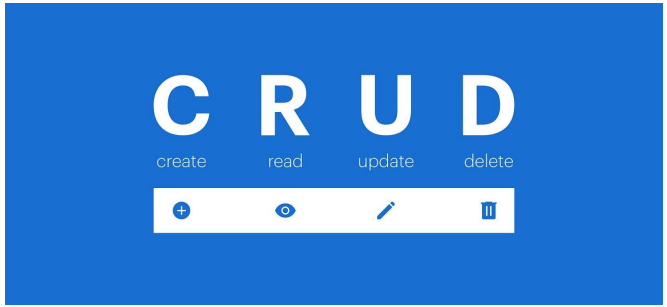
- **action**: Url a la que enviaremos los datos de formulario
- **method**: La forma que enviamos los datos al servidor:
  - **GET**: Los datos se envían junto a la URL y son visibles. No se usa cuando enviamos datos sensibles. Se suele usar con formularios de consulta de datos
  - **POST**: Los datos se envían de forma interna y no son visibles en la URL. Se usa para enviar datos sensibles. Se suele usar para crear, eliminar, editar, login,etc..
- **enctype**: indicamos el método que se usará para encriptar los datos en el envío:
  - **application/x-www-form-urlencoded:** Por defecto
  - **multipart/form-data :** Este método se usará cuando subamos archivos.
  - **text/plain**
- **input :** Elemento del formulario para introducir texto y enviarlo al servidor
- **label**: Etiqueta asociada a un input para dar información al usuario del dato a rellenar
- **button**: Botón que al pulsar ejecutamos el formulario y envía los datos a la url especificada.

CRUD

¿Qué es un **CRUD**?: CRUD es el acrónimo de **Create (Crear), Read (Leer), Update (Actualizar) y Delete (Borrar).**

Este concepto se utiliza para describir las cuatro operaciones básicas que pueden realizarse con un modelo en las aplicaciones Web.

Seguiremos usando los ejemplos de la Biblioteca para hacer el CRUD completo del modelo libro.



CRUD Índice

Create-Crear

Esta fase se utiliza para crear un nuevo registro en la base.

Para implementar la operación «Crear», es necesario proporcionar un formulario o una interfaz donde el usuario pueda ingresar los datos para el nuevo registro.

Después de que el usuario envía los datos, se debe realizar una validación de los mismos y luego insertarlos en la base o en el sistema de almacenamiento.

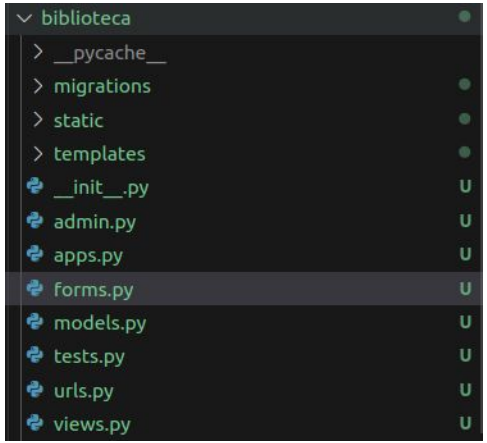
IMPORTANTE: LOS DATOS SE VALIDARÁN TANTO EN CLIENTE(CON JAVASCRIPT) COMO EN SERVIDOR(CON PYTHON Y DJANGO).:

- **CLIENTE: OFRECERÁ UNA PRIMERA CAPA DE SEGURIDAD Y FACILITARÁ MÁS AL USUARIO EL RELLENAR LOS DATOS,**
- **SERVIDOR: ES OBLIGATORIO AL 100% PARA ASEGURARNOS QUE LOS DATOS LLEGAN COMO QUEREMOS Y POR SEGURIDAD.**

Create-Crear

Ahora vamos a crear un formulario para añadir Libros en nuestra aplicación.

Para crear los formularios en Django, debemos crear primero la clase en nuestro proyecto que representará dicho formulario. Estas clases se irán creando en nuevo archivo que llamaremos **forms.py** y creamos dentro de nuestra aplicación.




```
from django import forms
```

Create-Crear

Ahora vamos a definir nuestro formulario para el modelo Libro:

```
class Libro(models.Model):
    IDIOMAS = [
        ("ES", "Español"),
        ("EN", "Inglés"),
        ("FR", "Francés"),
        ("IT", "Italiano"),
    ]

    nombre = models.CharField(max_length=200)
    idioma = models.CharField(
        max_length=2,
        choices=IDIOMAS,
        default="ES",
    )

    descripcion = models.TextField()
    fecha_publicacion = models.DateField()
    biblioteca = models.ForeignKey(Biblioteca, on_delete = models.CASCADE,related_name="libros_biblioteca")
    autores =   models.ManyToManyField(Autor,related_name="libros_autores")
```

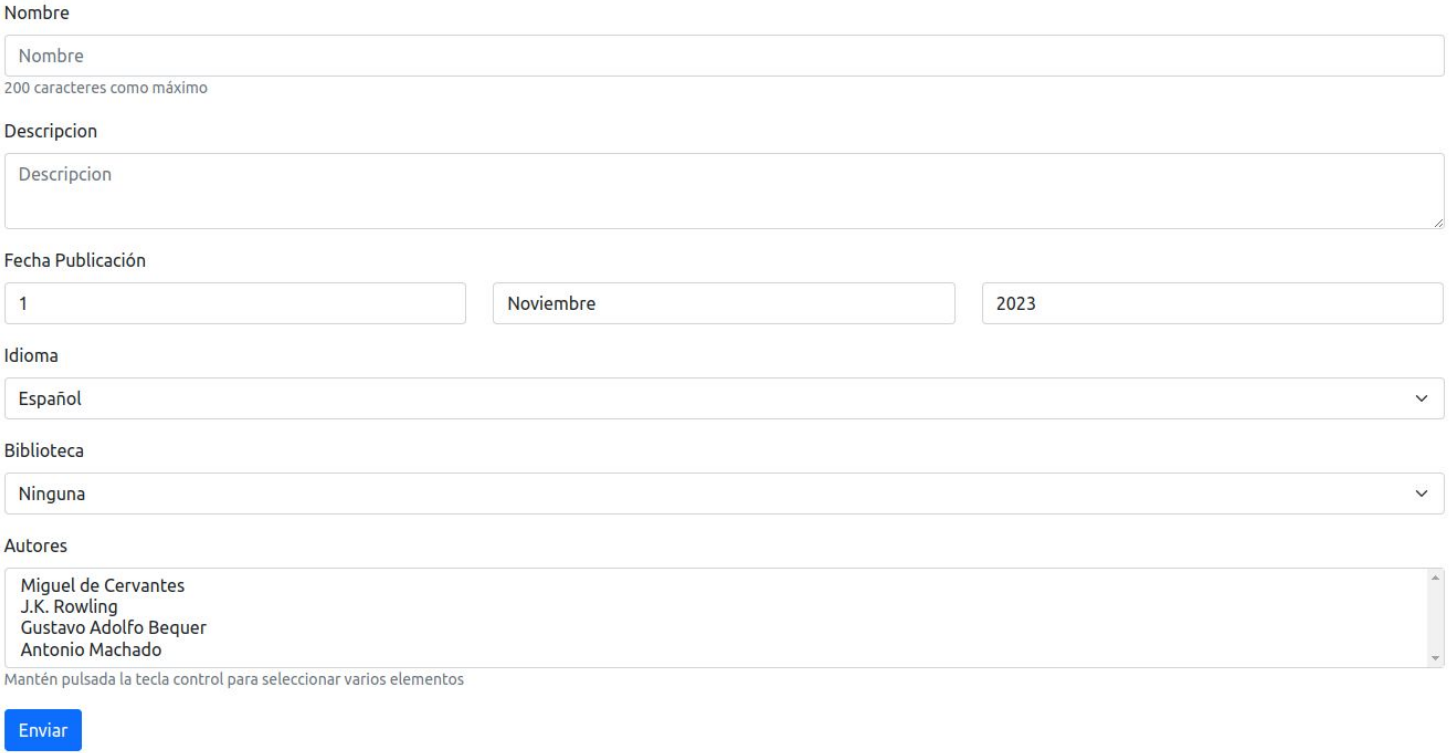
Formularios Genéricos

Para crear nuestro formulario usaremos campos específicos de la clase Forms de Django.

A continuación detallaremos en qué consiste cada parte del formulario

```python
class LibroForm(forms.Form):
    #Definimos un campo de tipo Texto para el nombre
    nombre = forms.CharField(label="Nombre",
                             required=True,
                             max_length=200,
                             help_text="200 caracteres como máximo")

    #Definimos un campo de Tipo Textarea para la descripcion
    descripcion = forms.CharField(label="Descripcion",
                                  required=False,
                                  widget=forms.Textarea())

    #Definimos un campo de Tipo Fecha para la fecha de publicación
    fecha_publicacion = forms.DateField(label="Fecha Publicación",
                                        initial=datetime.date.today,
                                        widget=forms.SelectDateWidget()
                                        )

    #Definimos un campo Select para seleccionar el Idioma
    idioma = forms.ChoiceField(choices=Libro.IDIOMAS,
                               initial="ES")

    #Definimos un campo Select para seleccionar una biblioteca que es una relacion ManyToOne
    bibliotecasDisponibles = Biblioteca.objects.all()
    biblioteca = forms.ModelChoiceField(
            queryset=bibliotecasDisponibles,
            widget=forms.Select,
            required=True,
            empty_label="Ninguna"
    )

     #Definimos un campo Select Múltiple para seleccionar autores en una relación ManyToMany
    autoresDisponibles = Autor.objects.all()
    autores = forms.ModelMultipleChoiceField(
        queryset= autoresDisponibles,
        required=True,
        help_text="Mantén pulsada la tecla control para seleccionar varios elementos"
    )
```

Formularios Genéricos



Formulario HTML Generado (Este formulario ha sido generado con la librería de estilo Bootstrap 5)

Formularios Genéricos

Empecemos por los **Forms Fields** que nos permitirá crear campos específicos de HTML**:**

- **forms.CharField:** Sería como un InputText en HTML y los usaremos para los campos tipo Char y Text
- **forms.DateField:** Sería como input tipo Date en HTML y lo usaremos para los campos tipo Fecha
- **forms.ChoiceField:** Sería como un select en HTML y lo usaremos para los campos tipo Char con opciones
- **forms.ModelChoiceField:** Sería como un Select en HTML y lo usaremos para las relaciones **OneToOne** o **ManyToOne**
- **forms.ModelMultipleChoiceField:** Sería como un Select Múltiple en HTML y lo usaremos para las relaciones **ManyToMany** o **OneToMany**
- Existen más tipos de campos que podemos encontrar en la documentación

Formularios Genéricos

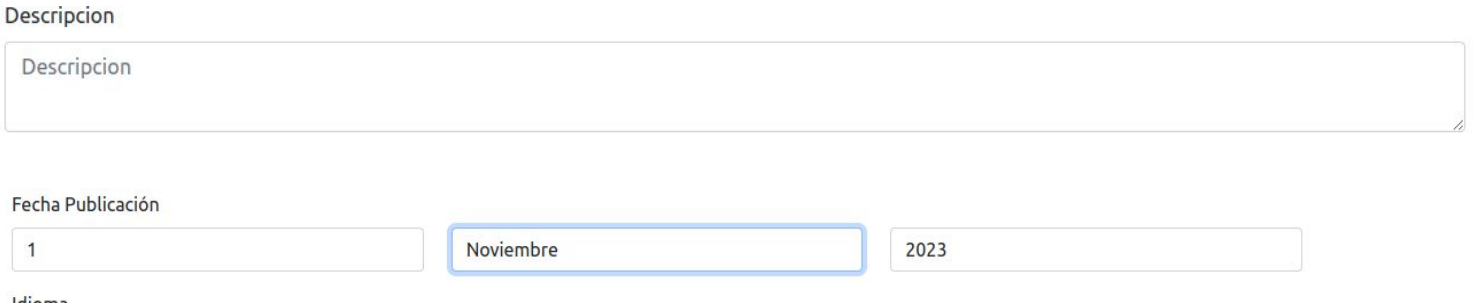
Los campos anteriormente especificados pueden tener una serie de parámetros para personalizarlos. :

- **required**: Para indicar si el campo es obligatorio en el formulario.
- **max_length**: Para indicar el tamaño máximo de caracteres del campo
- **help_text**: Texto de ayuda que aparecerá al lado del campo del formulario para facilitar la introducción de datos al usuario
- **label**: La etiqueta label que queremos que aparezca en el formulario asociada al campo
- **initial**: Valor por defecto en el campo
- **choices:** Para especificar los valores que aparecerán para seleccionar en el Select
- **empty_label**: Cuando hay que seleccionar en un Select
- **querySet**: QuerySet del Modelo correspondiente a la relación. Para que aparezcan las opciones correspondientes.

Podemos encontrar más parámetros en la documentación

Formularios Genéricos

Destacar los **widgets**: Son Clases de Django con sus **Templates** correspondiente que sirven para generar un elemento de formulario concreto. En los casos anteriores hemos usado uno para crear **Textarea** y otro para generar un **Selector de Fecha.** Para acceder a todos los widgets disponibles acceda a la **documentación**



Formularios Genéricos

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

Vista

```python
def libro_create(request):
    formulario = LibroForm()
    return render(request, 'libro/create.html',{"formulario":formulario})
```

Template

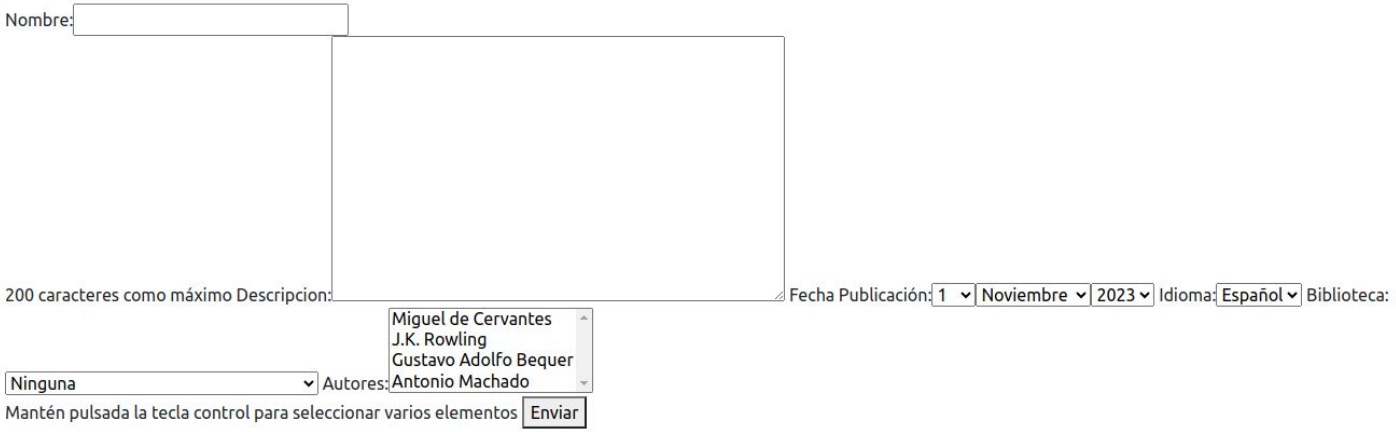
```
<form action = "" method = "post">

    {% csrf_token %}

    {{ formulario }}
    <button type="submit">Enviar</button>
</form>
```

Sólo debemos mostrar el valor de la variable.

IMPORTANTE TEMPLATE TAG {% csrf_token %}: Sirve para dotar de seguridad al formulario contra ataques Cross-Site Request Forgery (CSRF)

Formularios: Presentación

¡Problema que tenemos ahora!: EL ESTILO ES UN DESASTRE. Tenemos que darle estilo



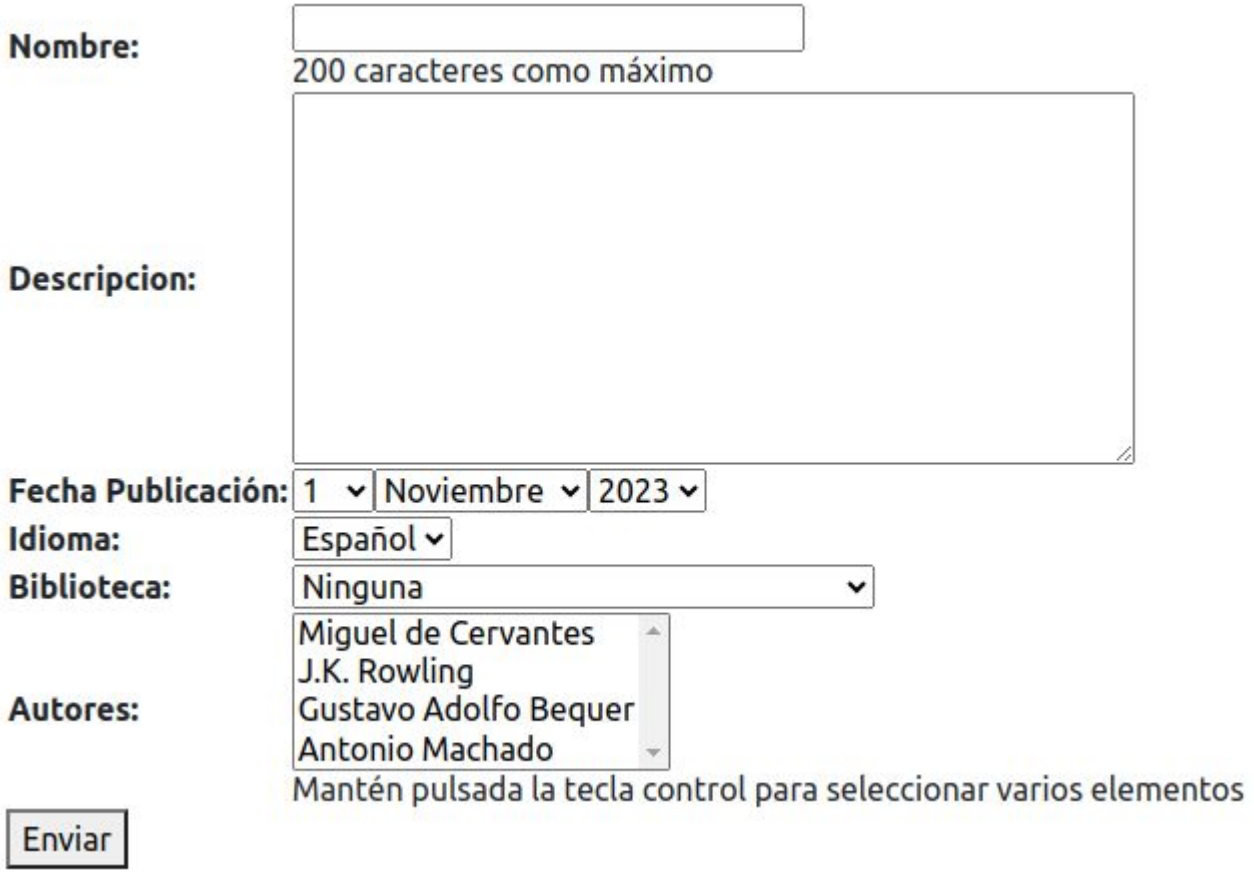
Formularios: Presentación

Por defecto Django tiene tres funciones dentro de la Clase **forms**, que permite formatear los formularios de diversas formas:

- **{{ form.as_table }}** lo creará como una tabla y cada campo lo incluye en una fila **<tr>**
- **{{ form.as_p }}** creará cada campo como un párrafo con **<p>**
- **{{ form.as_ul }}** cada campo lo agrupa en un elemento de lista con **<li>**

Formularios: Presentación

```
{{ form.as_table }}
```



Formularios
Presentación

{{ form.as_p }}

Nombre: 200 caracteres como máximo

Descripcion:

Fecha Publicación: 1 Noviembre 2023

Idioma: Español

Biblioteca: Ninguna

Autores: Miguel de Cervantes
J.K. Rowling
Gustavo Adolfo Bequer
Antonio Machado
Mantén pulsada la tecla control para seleccionar varios elementos

Enviar

Formularios Presentación

{{ form.as_ul }}

- Nombre: 200 caracteres como máximo
- Descripcion:
- Fecha Publicación: 1 Noviembre 2023
- Idioma: Español
- Biblioteca: Ninguna
- Autores: Miguel de Cervantes, J.K. Rowling, Gustavo Adolfo Bequer, Antonio Machado — Mantén pulsada la tecla control para seleccionar varios elementos

Enviar

Formularios:Presentación

¿Y cuál nos quedamos de las tres opciones?

NINGUNA

Las tres opciones nos dan una forma rápida de crear el formulario en la plantilla, y si usamos el css adecuado podemos darle un estilo que mejorar la presentación.

Pero en nuestro caso debemos aprender a aplicar los conocimientos aprendidos en Diseño de Interfaces, por esa razón crearemos los formularios manualmente.

Eso sí, usaremos el framework **Bootstrap 5** para dar estilos a nuestro proyecto. Por lo tanto debemos instalarlo en nuestro proyecto

Formularios:Bootstrap-V5

Vamos a instalar **Bootstrap5**:

- Documentación
- En el archivo requirements.txt incluimos: **django-bootstrap5~=24.3 y django-bootstrap-icons~=0.8.6**
- Ejecutamos el comando: **pip install -r requirements.txt**
- En **settings.py** dentro de la variable **INSTALLED_APPS** añadimos: **django_bootstrap5 y '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 %}**

Formularios:Bootstrap-V5

Ahora crearemos nuestro formulario manualmente:

Empezamos por el campo nombre y analizamos cada parte

```
<div class="mb-4">
    <label for="{{ formulario.nombre.id_for_label }}" class="form-label">{{ formulario.nombre.label }}: </label>
    <input  name="{{ formulario.nombre.html_name }}"
            type="text" class="form-control"
            id="{{ formulario.nombre.id_for_label }}"
            maxlength="{{ formulario.nombre.field.max_length }}"
            {% if formulario.nombre.field.required %}
                required
            {% endif %}
    />
    {% if formulario.nombre.help_text %}
    <div class="form-text">
        {{ formulario.nombre.help_text }}
    </div>
    {% endif %}
</div>
```

Formularios:Bootstrap-V5

Explicamos los **tags** que hemos incluido en la plantilla:

- **{{ formulario.nombre.id_for_label }}** : Para usar el identificador del input.
- **{{ formulario.nombre.label }}** : Para indicar el nombre del label
- **{{ formulario.nombre.html_name }}** : Para indicar el nombre del campo del formulario (Este es importantísimo porque es el valor que recibiremos en las vistas a posteriori)
- **{{ formulario.nombre.field.max_length }}** : Para indicar la validación de máximo caracteres
- **formulario.nombre.field.required:** Para comprobar si el campo es obligatorio o no.
- **{{ formulario.nombre.help_text }} :** Para el texto de ayuda del campo

Formularios:Bootstrap-V5

Campo Descripción:

```
<div class="mb-4">
    <label for="{{ formulario.descripcion.id_for_label }}" class="form-label">{{ formulario.descripcion.label }}: </label>
    <textarea name="{{ formulario.descripcion.html_name }}" class="form-control" id="{{ formulario.descripcion.id_for_label }}"></textarea>
</div>
```

Campo Fecha:

```
<div class="mb-4">
    {% bootstrap_field formulario.fecha_publicacion show_label=True %}
</div>
```

Formularios:Bootstrap-V5

En el campo de **Descripción** no hemos incluido nada nuevo, pero el campo **fecha publicación** si es bastante distinto, es porque hemos usado la librería Bootstrap para facilitarnos la creación del campo:

- Para que funcione debemos cargar la librería Bootstrap5 en la plantilla añadiendo: **{% load bootstrap5 %}**
- Para usarlo, tenemos que utilizar la tag **bootstrap_field nombre_del_campo parametros:**

```
{% bootstrap_field formulario.fecha_publicacion show_label=True %}
```

- Podemos encontrar información de todos los parámetros en la documentación
- En los parámetros podemos incluir clases de estilos específicas para el campo.

Formularios:Bootstrap-V5

Aunque ya hemos aprendido a usar bootstrap para generar nuestros campos, voy a explicar otro ejemplo para un select sin usar la librería de bootstrap.

Destacamos como se recorre los valores usando el atributo **subwidgets** que es quién contiene los valores de **choices** del campo

```
<div class="mb-4">
    <label class="form-label" for="{{ formulario.idioma.id_for_label }}">
        {{formulario.idioma.label}}
    </label>
    <select name="{{ formulario.idioma.html_name }}"
    value="{{ formulario.idioma.value }}"
    id="{{ formulario.idioma.id_for_label }}"
    class="form-select" >
        {% for widget in formulario.idioma.subwidgets %}
            {{ widget }}
        {% endfor %}
    </select>
</div>
```

Formularios:Bootstrap-V5

Por último creamos los dos últimos campos usando la librería de Bootstrap 5:

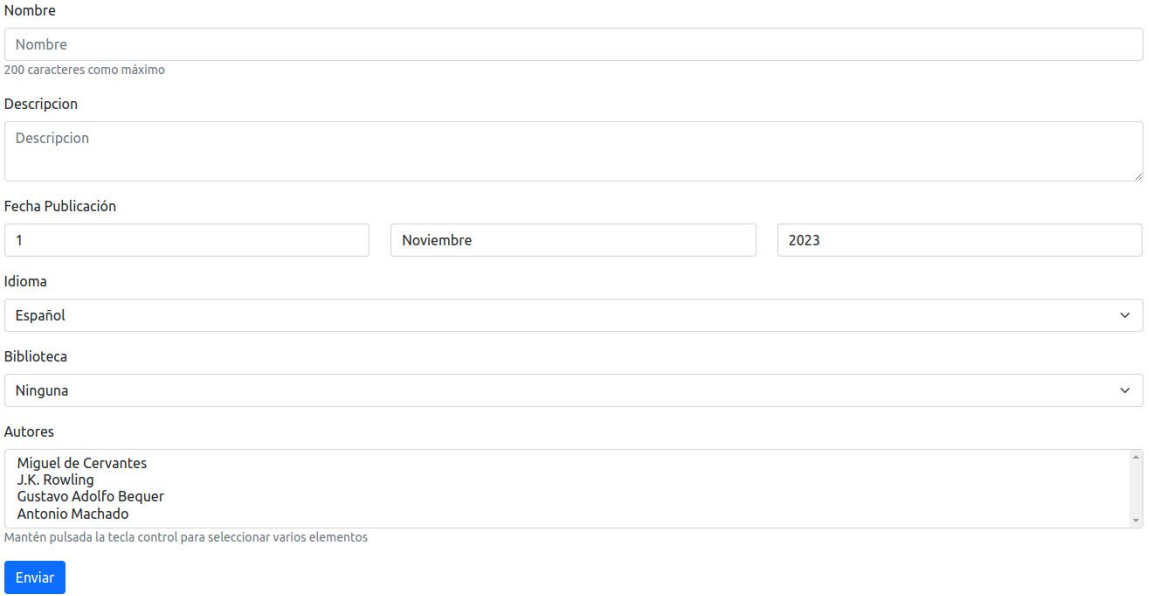
```
<div class="mb-4">
    {% bootstrap_field formulario.biblioteca show_label=True %}
</div>

<div class="mb-4">
    {% bootstrap_field formulario.autores show_label=True %}
</div>
```

IMPORTANTE: El uso de la librería de Bootstrap nos facilita el estilo de la aplicación, pero a la vez nos limita nuestra propia personalización del Estilo.

Formularios:Bootstrap-V5

Por último ,en tema de presentación de formularios, con una simple línea podríamos haber creado el formulario con estilo de bootstrap:

```
{% bootstrap_form formulario %}
```



Formularios: Model Forms

Una vez visto los formularios genérico , ahora vamos a ver otras formas de crear formularios de una forma más rápida y cómoda para crear los formularios. Pero son menos personalizables que los genéricos.

Los **Models Forms.**

```
from django.forms import ModelForm
```

```
class LibroModelForm(ModelForm):
    class Meta:
        model = Libro
        fields = ['nombre','descripcion','fecha_publicacion','idioma','biblioteca','autores']
        labels = {
            "nombre": ("Nombre del Libro"),
        }
        help_texts = {
            "nombre": ("200 caracteres como máximo"),
            "autores":("Mantén pulsada la tecla control para seleccionar varios elementos")
        }
        widgets = {
            "fecha_publicacion":forms.SelectDateWidget()
        }
        localized_fields = ["fecha_publicacion"]
```

Formularios: Model Forms

Estos formularios(**ModelForm**) son más específicos de un Modelo en concreto, mientras que los formularios genéricos (**forms.Form**), nos da más libertad para personalizar nuestro formulario mezclando Modelos diferentes u otros campos.

- Documentación
- El campo fields sirve para especificar los campos que queremos que aparezca en el formulario, si queremos todos, usamos la palabra clave: '__all__'
- Los demás atributos usan diccionarios para indicar que parámetros necesitas por cada atributo.Todos los parámetros posibles aparecen en la documentación, pero tienen una similitud a los usados en los formularios genéricos.
- Destacar el campo localized_fields para indicar los campos que debemos tener en cuenta que deben adaptarse a la localización de la ciudad y el país, como por ejemplo las fechas y horas.

Formularios:Model Form Factory

Model Form Factory: otra forma más de crear Formularios, que no necesitamos ni incluirlo en el archivo **forms.py.** Sólo crearlo en las vistas

```
from django.forms import modelform_factory
```

```
formularioLibroFactory = modelform_factory(Libro,
                                           fields='__all__',
                                           widgets = {
                                                "fecha_publicacion":forms.SelectDateWidget()
                                           })
formulario = formularioLibroFactory()
```

Los atributos para crear el formulario son los mismos que los Models Forms

Formularios

¿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.

Formularios

Repasemos:

- Hemos creado la URL y La vista que gestionar el Formulario de Crear Libro
- Hemos creado el formulario para añadir Libros
- Hemos creado la plantilla con el HTML para crear Libros.

¿Qué nos toca ahora?

Procesar en la vista los datos que enviaremos desde el formulario y crear el Libro en la base datos.

Create-Crear

Ahora nos situamos en las vistas.

Lo primero que tenemos que hacer es diferenciar cuando la petición va por **GET** o va por **POST.**

- **GET:** Debemos crear el formulario con los datos vacíos. Por lo tanto creamos el formulario del Modelo y se lo pasamos a la Plantilla:
- **POST:** Debemos seguir los siguientes pasos:
    1. Creamos el formulario con los datos que enviamos
    2. Comprobamos si son válidos los datos del formulario. Si son válidos continuamos, sino volvemos a la página con el formulario indicando los errores
    3. Creamos un Objeto del Modelo Libro
    4. Incluimos las relaciones ManyToMany
    5. Guardamos el modelo en la base de datos
    6. Redireccionamos a la URL que muestre todos los libros

Create Crear:Formularios Genéricos

```python
def libro_create(request):

    # 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 = LibroForm(datosFormulario)

    if (request.method == "POST"):
        # Llamamos la función que creará el libro
        libro_creado = crear_libro_generico(formulario)
        if(libro_creado):
            return redirect("lista_libros")

    return render(request, 'libro/create.html',{"formulario":formulario})
```

Create Crear:Formularios Genéricos

```python
def crear_libro_generico(formulario):
    libro_creado = False
    # Comprueba si el formulario es válido
    if formulario.is_valid():

        # Obtiene los datos del formulario, y los limpia de espacios blancos y carácteres raros.
        libro = Libro.objects.create(
                nombre = formulario.cleaned_data.get('nombre'),
                idioma = formulario.cleaned_data.get('idioma'),
                descripcion = formulario.cleaned_data.get('descripcion'),
                fecha_publicacion = formulario.cleaned_data.get('fecha_publicacion'),
                biblioteca = formulario.cleaned_data.get('biblioteca'),
        )

        #Añade los autores que son relaciones ManyToMany
        libro.autores.set(formulario.cleaned_data.get('autores'))
        try:
            # Guarda el libro en la base de datos
            libro.save()
            libro_creado = True
        except:
            pass
    return libro_creado
```

Create Crear: Formulario Modelos

Creamos el formulario de tipo ModelForm

```
formulario = LibroModelForm(datosFormulario)
```

El código para crear el Libro es más sencillo, es simplemente guardar:

```
def crear_libro_modelo(formulario):
    libro_creado = False
    # Comprueba si el formulario es válido
    if formulario.is_valid():
        try:
            # Guarda el libro en la base de datos
            formulario.save()
            libro_creado = True
        except:
            pass
    return libro_creado
```

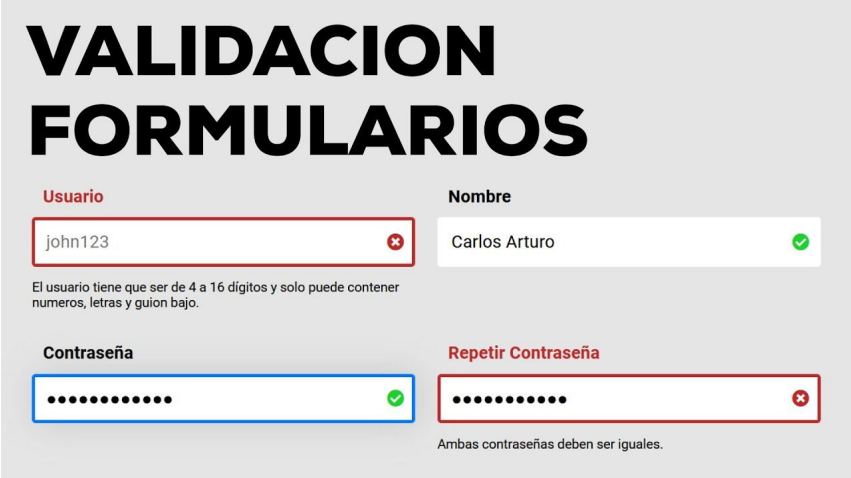
Create(crear):Formulario Modelos

La forma de procesar este formulario en **views.py:**

```
def libro_create(request):

    # 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 = LibroModelForm(datosFormulario)

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

        libro_creado = crear_libro_modelo(formulario)
        if(libro_creado):
            messages.success(request, 'Se ha creado el libro'+formulario.cleaned_data.get('nombre')+" correctamente")
            return redirect("libro_lista")

    return render(request, 'libro/create.html',{"formulario":formulario})
```

Formularios: Validación en Servidor

Por último nos queda un apartado bastante importante que no hemos saltado. Y ahora veremos en profundidad



Formularios: Validación en Servidor

¿Qué es la validación en Servidor?

Es el proceso por el cual se comprueba si los datos que se han enviado al servidor son totalmente correctos y pueden añadirse a la base de datos sin problema.

Aunque en cliente hagamos una validación, es importante volver a hacerla en el servidor, porque en cliente, el usuario puede evadirla, pero la validación en servidor no puede evitarla.

Normalmente cuando se valida en servidor, si hay errores, la aplicación Web vuelve al formulario, para indicar, marcar y mostrar los errores en el formulario, para que el usuario entienda qué está mal, para corregirlo.

Formularios: Validación en Servidor

A continuación vamos a incluir una validación para cada uno de los campos:

- El campo nombre no exista ya en la base de datos
- El mínimo de caracteres del campo descripción sea 10
- La fecha de publicación sea menor que hoy
- Que el idioma no pueda ser en Francés si se ha seleccionado la Biblioteca de la Universidad de Sevilla
- Que al menos seleccione dos autores

Formularios: Validación en Servidor

Para añadir validaciones a los formularios debemos cambiar nuestra clase formulario y añadir un nuevo método: **clean**

```
def clean(self):

    super().clean()
```

Este método sirve para comprobar si los campos tienen el valor adecuado para el modelo. Por lo tanto lo primero que debemos hacer antes de hacer validaciones personalizadas, es comprobar que los datos nos hayan llegado bien, llamando a la función padre.

Formularios: Validación en Servidor

El siguiente paso es obtener todos los valores de los campos validados, estos datos están almacenados es una lista que se llama **cleaned_data** del objeto **formulario**

```
#Obtenemos los campos
nombre = self.cleaned_data.get('nombre')
descripcion = self.cleaned_data.get('descripcion')
fecha_publicacion = self.cleaned_data.get('fecha_publicacion'
idioma = self.cleaned_data.get('idioma')
biblioteca = self.cleaned_data.get('biblioteca')
autores = self.cleaned_data.get('autores')
```

Formularios: Validación en Servidor

Y ahora realizamos las validaciones:

```
#Comprobamos que no exista un libro con ese nombre
libroNombre = Libro.objects.filter(nombre=nombre).first()
if(not libroNombre is None ):
    self.add_error('nombre','Ya existe un libro con ese nombre')
```

Cada vez que se produzca un error, se debe añadir el error con el método **add_error** , el primer parámetro es el campo que tiene el error y el segundo parámetro es el error.

Un mismo campo puede tener varios errores. Por eso deben añadirse de esta forma.

Formularios: Validación en Servidor

A continuación muestro las otras validaciones:

```
#Comprobamos que el campo descripción no tenga menos de 10 caracteres
if len(descripcion) < 10:
    self.add_error('descripcion','Al menos debes indicar 10 caracteres')

#Comprobamos que la fecha de publicación sea mayor que hoy
fechaHoy = date.today()
if fechaHoy < fecha_publicacion :
    self.add_error('fecha_publicacion','La fecha de publicacion debe ser mayor a Hoy')

#Comprobamos que el idioma no pueda ser en Francés si se ha seleccionado la Biblioteca de la Universidad de Sevilla
if idioma == "FR" and biblioteca.id == 3:
    self.add_error('idioma','No puede usar la Biblioteca de la Universidad de Sevilla y el idioma Fránces')
    self.add_error('biblioteca','No puede usar la Biblioteca de la Universidad de Sevilla y el idioma Fránces')

#Que al menos seleccione dos autores
if len(autores) < 2:
    self.add_error('autores','Debe seleccionar al menos dos autores')
```

Formularios: Validación en Servidor

Por último debemos especificar que devuelve el método:

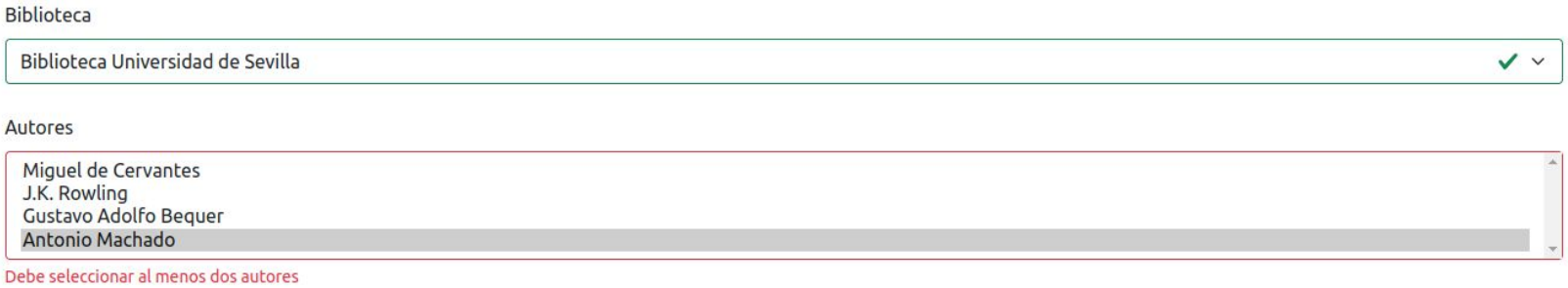
```
#Siempre devolvemos el conjunto de datos.
return self.cleaned_data
```

Se devuelve siempre el conjunto de datos validados. Si contiene errores la función is_valid() devolverá False.

Formularios: Validación en Servidor

¿Cómo mostramos esos errores en el Formulario?:

En el caso de haber usado la librería de Bootstrap, no debemos preocuparnos porque lo hará sólo, y el resultado será como el siguiente:



Formularios: Validación en Servidor

Pero en el caso de los campos que hemos personalizados el estilo sin librería si tenemos que modificar la Template y gestionar los errores de la siguiente forma:

- En el atributo class del campo, comprobar si el campo tiene errores de validación, si los tiene añadimos la clase **is-invalid,** en el caso de que no tenga errores, debemos comprobar si hemos enviado el formulario. Para ello tenemos la propiedad **is_bound,** que es **True** si el formulario viene de una validación o **False** si se ha cargado por una URL.
- Esto servirá para marcar en rojo el campo correspondiente en el formulario.

```
class="form-control {% if formulario.nombre.errors %} is-invalid{% elif formulario.is_bound %} is-valid{% endif %}"
```

Formularios: Validación en Servidor

Es importante mostrar el valor del campo que hemos enviado y que no aparezca vacío cuando tengamos errores, para ello usamos lo siguiente:

```
value="{% if formulario.is_bound %}{{ formulario.nombre.value }}{% endif %}"
```

Volvemos a usar la propiedad **is_bound,** porque la primera vez que accedemos el campo estará vacío, pero sí hemos enviado el dato al servidor y es erróneo, si debería aparecer lo que enviamos

Formularios: Validación en Servidor

Por último, debemos mostrar los errores, para ello, cada campo tiene un listado de los errores, sólo tenemos que recorrer esta lista de errores y mostrarlos uno a uno dentro de un div con un estilo en concreto:

```
<div class="invalid-feedback">
    {% for error in formulario.nombre.errors %}
        {{ error }}
    {% endfor %}
</div>
```

* Puede que existan casos de Widgets específicos como el de Fecha, que tendremos que incluir esta parte también para mostrar los errores.

Read(Retrieve)-Leer

Esta fase se utiliza para leer los datos de la base y mostrarlos al usuario.

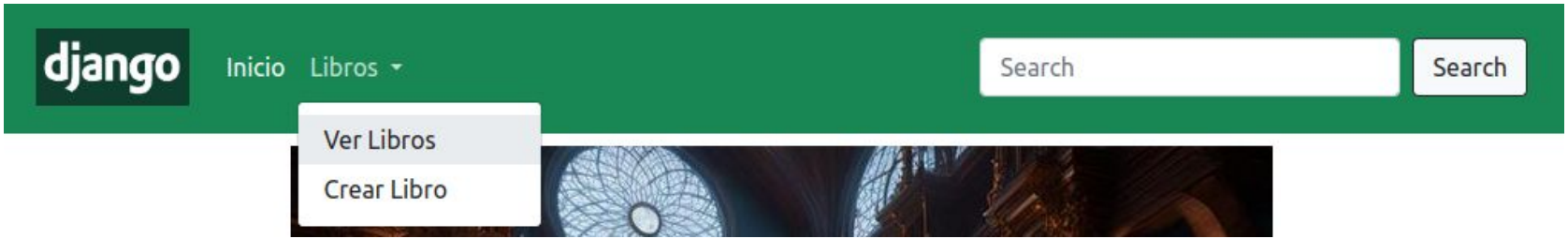
Se debe proporcionar al usuario una interfaz que permita al usuario buscar y recuperar los registros existentes.

El temas anteriores vimos las Urls, Views y QuerySet para obtener datos de las base de datos y mostrarla. Esa es la parte de “Leer”. Ejemplos que hemos visto:

- Mostrar un listado de libros
- Mostrar información de un libro en concreto
- Cuadro de búsqueda
- Filtros de búsqueda

Read(Retrieve)-Leer

Por ejemplo a continuación tenemos un ejemplo de menú que tiene enlaces a las páginas para ver todos los libros o crear un libro. Además tiene una barra de búsqueda para buscar por todos los libros.



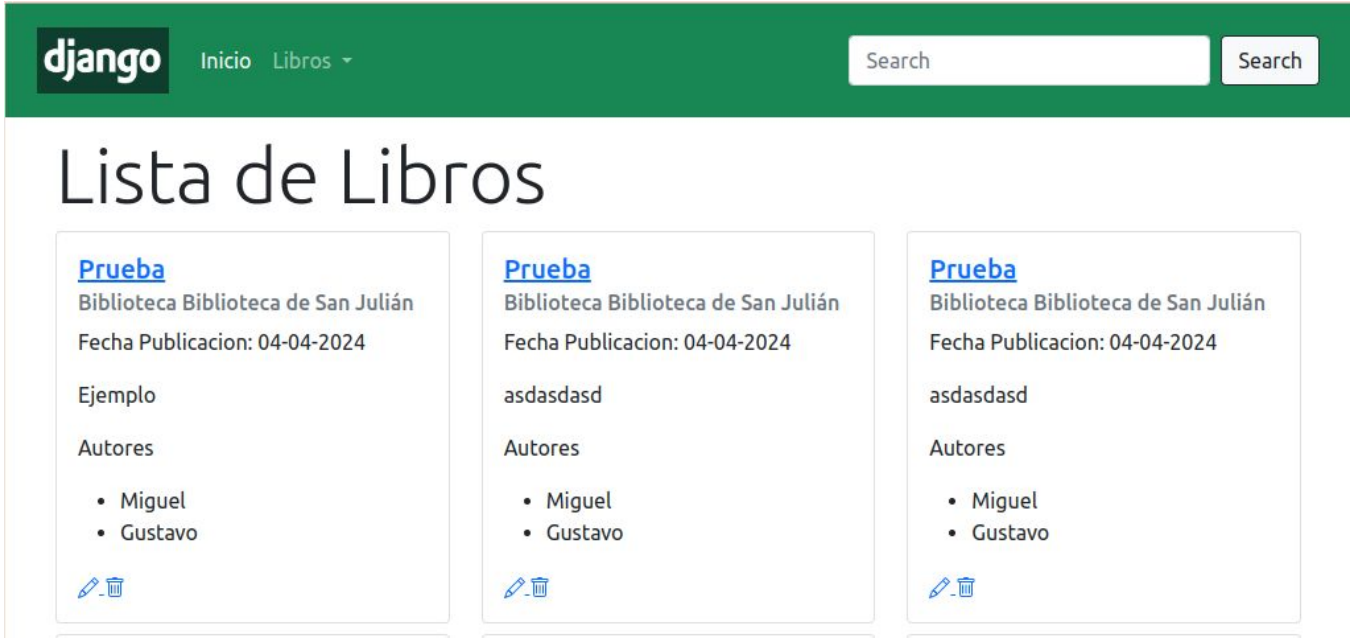
Read(Retrieve)-Leer

Los links de las urls se crean como hemos visto en temas anteriores:

```
<li><a class="dropdown-item" href="{% url 'lista_libros' %}">Ver Libros</a></li>
<li><a class="dropdown-item" href="{% url 'libro_create' %}">Crear Libro</a></li>
```

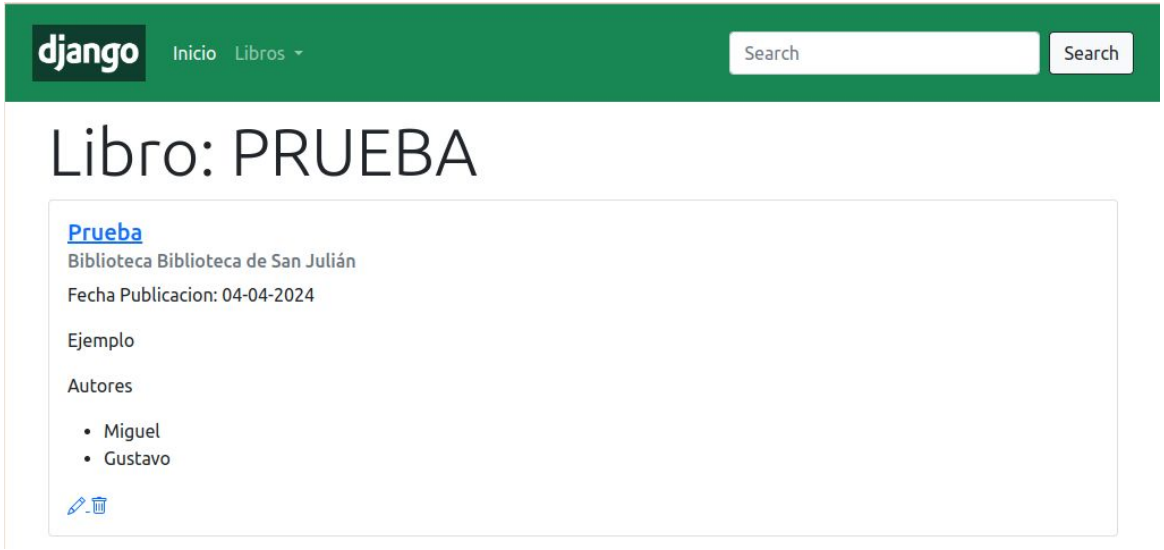
Read(Retrieve)-Leer

Las urls muestran páginas de ejemplo, como las siguientes:



Read(Retrieve)-Leer

Mostramos un libro en concreto:



Read(Retrieve)-Leer

A continuación explicaré como crear el formulario de búsqueda, para encontrar todos los libros que tengan en el título o en el contenido.

1º Creamos la Url

```
path('libro/buscar/',views.libro_buscar,name='libro_buscar'),
```

2º Creamos el Formulario

IMPORTANTE: El formulario solo tendrá un campo y este no está relacionado con ningún modelo, por esa razón usamos un formulario genérico

```
class BusquedaLibroForm(forms.Form):
    textoBusqueda = forms.CharField(required=True)
```

Read(Retrieve)-Leer

3º Creamos el formulario HTML de Búsqueda:

```
<form action="{% url 'libro_buscar' %}" class="d-flex mt-3" role="search" method="GET">
  <input name="textoBusqueda" class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
  <button class="btn btn-light btn-outline-dark" type="submit">Search</button>
</form>
```

¿Qué tenemos distinto?

Este formulario es de Tipo **GET** por lo tanto el texto de búsqueda que introducimos, aparecerá en la URL.

127.0.0.1:8000/libro/buscar/?textoBusqueda=Ejemplo

Read(Retrieve)-Leer

4º Creamos la vista:

```
def libro_buscar(request):

    formulario = BusquedaLibroForm(request.GET)

    if formulario.is_valid():
        texto = formulario.cleaned_data.get('textoBusqueda')
        libros = Libro.objects.select_related("biblioteca").prefetch_related("autores")
        libros = libros.filter(Q(nombre__contains=texto) | Q(descripcion__contains=texto)).all()
        return render(request, 'libro/lista_busqueda.html',{"libros_mostrar":libros,"texto_busqueda":texto})
    if("HTTP_REFERER" in request.META):
        return redirect(request.META["HTTP_REFERER"])
    else:
        return redirect("index")
```

¿Qué destacamos?

- En este caso los datos del formulario lo obtenemos por GET
- Si el formulario no es correcto(Sólo comprobamos que haya introducido un valor), quiero que me redireccione a la página desde la que hice la búsqueda, para ello usamos el atributo: **request.META["HTTP_REFERER"]**
- Debemos controlar antes si existe la propiedad, por si no proviene desde otra dirección de la página WEB, sino se ha introducido directamente en la URL.
- **request** tiene muchos atributos que nos pueden ser de utilidad: Documentación

Read(Retrieve)-Leer

A continuación explicaré como crear el formulario de búsqueda avanzada:

- Buscar por texto
- Buscar por Idioma
- Buscar por fecha

Read(Retrieve)-Leer

Lo primero es crear el formulario:

¿Qué Destacamos?

- Usamos **MultipleChoiceField** para los idiomas, porque podemos filtrar por varios en la búsqueda
- El Widget **SelectDateWidget**, podemos especificar el rango de años. Por defecto siempre es desde el año actual hasta 9 más.

```
class BusquedaAvanzadaLibroForm(forms.Form):

    textoBusqueda = forms.CharField(required=False)


    idiomas = forms.MultipleChoiceField(choices=Libro.IDIOMAS,
                                        required=False,
                                        widget=forms.CheckboxSelectMultiple()
                                       )

    fecha_desde = forms.DateField(label="Fecha Desde",
                                  required=False,
                                  widget= forms.SelectDateWidget(years=range(1990,2023))
                                  )

    fecha_hasta = forms.DateField(label="Fecha Desde",
                                    required=False,
                                    widget= forms.SelectDateWidget(years=range(1990,2023))
                                    )
```

Read(Retrieve)-Leer

Ahora toca crear la validación.
Teniendo en cuenta lo siguiente:

- Ningún campo es obligatorio, pero al menos debe introducir un valor en alguno para buscar
- Si introduce un texto al menos que tenga 3 caracteres o más
- La fecha hasta debe ser mayor o igual a fecha desde. Pero sólo se valida si han introducido ambas fechas

```
def clean(self):

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

    #Obtenemos los campos
    textoBusqueda = self.cleaned_data.get('textoBusqueda')
    idiomas = self.cleaned_data.get('idiomas')
    fecha_desde = self.cleaned_data.get('fecha_desde')
    fecha_hasta = self.cleaned_data.get('fecha_hasta')

    #Controlamos los campos
    if(textoBusqueda == ""
       and len(idiomas) == 0
       and fecha_desde is None
       and fecha_hasta is None
       ):
        self.add_error('textoBusqueda','Debe introducir al menos un valor en un campo del formulario')
        self.add_error('idiomas','Debe introducir al menos un valor en un campo del formulario')
        self.add_error('fecha_desde','Debe introducir al menos un valor en un campo del formulario')
        self.add_error('fecha_hasta','Debe introducir al menos un valor en un campo del formulario')
    else:
        if(textoBusqueda != "" and len(textoBusqueda) < 3):
            self.add_error('textoBusqueda','Debe introducir al menos 3 caracteres')

        if(not fecha_desde is None  and not fecha_hasta is None and fecha_hasta < fecha_desde):
            self.add_error('fecha_desde','La fecha hasta no puede ser menor que la fecha desde')
            self.add_error('fecha_hasta','La fecha hasta no puede ser menor que la fecha desde')

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

Read(Retrieve)-Leer

Ahora toca crear la vista. Iremos mostrando cada parte:

```
def libro_buscar_avanzado(request):

    if(len(request.GET) > 0):
        formulario = BusquedaAvanzadaLibroForm(request.GET)
        if formulario.is_valid():…
    else:
        formulario = BusquedaAvanzadaLibroForm(None)
    return render(request, 'libro/busqueda_avanzada.html',{"formulario":formulario})
```

Lo primero que debemos controlar es si recibimos parámetros por GET, esto es importante, porque al ser GET, las peticiones pueden llegarnos desde la URL en vez de un formulario, por lo tanto debemos saber si tenemos datos o no, para procesar el formulario y validarlo.

En el caso que procesa desde una URL y no tenga datos, mostramos el formulario correspondiente.

Read(Retrieve)-Leer

El siguiente paso es obtener los datos, y crear dos variables:

- Una variable **mensaje_busqueda** para tener un texto que indique los filtros aplicados
- La variable **QSlibros** que es la QuerySet con los filtros que se van a ir aplicando. Tened en cuenta que la QuerySet se irá creando dependiendo de los parámetros que le pasemos

```
mensaje_busqueda = "Se ha buscado por los siguientes valores:\n"

QSlibros = Libro.objects.select_related("biblioteca").prefetch_related("autores")

#obtenemos los filtros
textoBusqueda = formulario.cleaned_data.get('textoBusqueda')
idiomas = formulario.cleaned_data.get('idiomas')
fechaDesde = formulario.cleaned_data.get('fecha_desde')
fechaHasta = formulario.cleaned_data.get('fecha_hasta')
```

Read(Retrieve)-Leer

Comprobamos si obtenemos datos del texto búsqueda y aplicamos el filtro:

```
#Por cada filtro comprobamos si tiene un valor y lo añadimos a la QuerySet
if(textoBusqueda != ""):
    QSlibros = QSlibros.filter(Q(nombre__contains=texto) | Q(descripcion__contains=texto))
    mensaje_busqueda +=" Nombre o contenido que contengan la palabra "+texto+"\n"
```

Read(Retrieve)-Leer

Comprobamos si ha seleccionado un idioma y los incluimos en la QuerySet. Tened en cuenta que idioma es una lista de idiomas seleccionados, debemos iterar y crear un filtro con **Q y |(OR)**

```
#Si hay idiomas, iteramos por ellos, creamos la queryOR y le aplicamos el filtro
if(len(idiomas) > 0):
    mensaje_busqueda +=" El idioma sea "+idiomas[0]
    queryOR = Q(idioma=idiomas[0])
    for idioma in idiomas[1:]:
        mensaje_busqueda += " o "+idiomas[1]
        queryOR |= Q(idioma=idioma)
    mensaje_busqueda += "\n"
    QSlibros =  QSlibros.filter(queryOR)
```

Read(Retrieve)-Leer

Realizamos los filtros de fechas:

```
#Comprobamos fechas
#Obtenemos los libros con fecha publicacion mayor a la fecha desde
if(not fechaDesde is None):
    mensaje_busqueda +=" La fecha sea mayor a "+datetime.strftime(fechaDesde,'%d-%m-%Y')+"\n"
    QSlibros = QSlibros.filter(fecha_publicacion__gte=fechaDesde)

 #Obtenemos los libros con fecha publicacion menor a la fecha desde
if(not fechaHasta is None):
    mensaje_busqueda +=" La fecha sea menor a "+datetime.strftime(fechaHasta,'%d-%m-%Y')+"\n"
    QSlibros = QSlibros.filter(fecha_publicacion__lte=fechaDesde)
```

Read(Retrieve)-Leer

Por último, ejecutamos la querySet y enviamos los libros:

```
libros = QSlibros.all()

return render(request, 'libro/lista_busqueda.html',
                {"libros_mostrar":libros,
                 "texto_busqueda":mensaje_busqueda})
```

Read(Retrieve)-Leer

Ejemplo de Resultado:

Update-Actualizar

Esta fase se utiliza para actualizar los datos existentes en la base de datos.

Debemos proporcionar una interfaz para que el usuario pueda modificar los datos de un registro existente.

Una vez que el usuario envía los datos actualizados, debe realizarse una validación de los mismos y luego actualizar el registro correspondiente.

En Django,una vez hecho el formulario de Crear, el formulario de Editar es sencillo, porque sólo tenemos que utilizar el mismo pero con algunos cambios.

Update-Actualizar

Primero vamos con la URL:

```
path('libro/editar/<int:libro_id>',views.libro_editar,name='libro_editar'),
```

Es necesario pasarle el id del libro que vamos a editar.

Update-Actualizar

Ahora vamos con la vista:

```
def libro_editar(request,libro_id):
    libro = Libro.objects.get(id=libro_id)

    datosFormulario = None

    if request.method == "POST":
        datosFormulario = request.POST

    formulario = LibroModelForm(datosFormulario,instance = libro)

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

        if formulario.is_valid():
            formulario.save()
            try:
                formulario.save()
                return redirect('libro_lista')
            except Exception as e:
                pass
    return render(request, 'libro/actualizar.html',{"formulario":formulario,"libro":libro})
```

Es exactamente igual que la función de crear con ModelForms, pero al crear el formulario le pasamos el libro que vamos a editar, para que cargue los valores iniciales.

Update-Actualizar

Ahora vamos con la plantilla:

CREAMOS UNA PLANTILLA actualizar.html EXACTAMENTE IGUAL QUE LA DE CREATE, PERO CON DOS CAMBIOS:

1º Debemos indicar la nueva URL donde enviaremos los datos del formulario

```
<form action = "{% url 'libro_editar' libro.id %}" method = "post">
```

Update-Actualizar

2ºEl formulario tiene que tener cargado por defecto todos los datos del libro, por lo tanto además de rellenar el formulario cuando proviene de una validación, también tenemos que hacerlo a cargar el formulario :

Con comprobar que existe la variable libro es suficiente

```
value="{% spaceless %}
        {% if formulario.is_bound%}
            {{ formulario.nombre.value }}
        {% elif libro %}
            {{ formulario.nombre.value }}
        {% endif %}
        {% endspaceless %}"
```

Usamos la etiqueta **spaceless** para quitar los espacios en HTML que no queremos al formatear el código

Update-Actualizar

Ejercicio de mejora y optimización:

Si sólo nos ha hecho falta hacer dos modificaciones y el formulario es casi igual

¿Qué cambios podríamos hacer con los extends e include de las Templates para no repetir código?

Update-Actualizar

Por último falta cambiar un detalle al validar el formulario.

En nuestro proyecto, no puede existir dos libros con el mismo nombre, por lo tanto al ahora de editar un campo del libro que no sea el nombre, nos dará un error de que ya existe un libro con ese nombre en la base de datos.

En este caso sólo tenemos que controlar que si tenemos una instancia del Modelo y el id coincide con el libro encontrado por nombre, no añadamos el error.

```
#Comprobamos que no exista un libro con ese nombre
libroNombre = Libro.objects.filter(nombre=nombre).first()
if(not (libroNombre is None
        or (not self.instance is None and libroNombre.id == self.instance.id)
        )
    ):
    self.add_error('nombre','Ya existe un libro con ese nombre')
```

Delete-Eliminar

Esta fase se utiliza para eliminar un registro existente.

Debemos una interfaz que permita al usuario seleccionar un registro existente y **confirmar** su eliminación.

Es importante hacer énfasis en que el usuario debe confirmar esa eliminación, siempre es importante incluir un botón que el usuario pueda cancelar el proceso de eliminación antes de proceder a ella.

Delete-Eliminar

Primero creamos la URL:

Debemos pasarle el id del libro a eliminar

```
path('libro/eliminar/<int:libro_id>',views.libro_eliminar,name='libro_eliminar'),
```

Delete-Eliminar

Creamos la Vista:

```
def libro_eliminar(request,libro_id):
    libro = Libro.objects.get(id=libro_id)
    try:
        libro.delete()
    except:
        pass
    return redirect('libro_lista')
```

Solo tenemos que llamar al método **delete**.

Delete-Eliminar

Ahora debemos crear el formulario para eliminar.

```
<div class="mb-2">
    <form action = "{% url 'libro_eliminar' libro.id %}" method = "post">
        {% csrf_token %}
        <button class="btn btn-danger" onclick="return eliminar();" type="submit">
            <span>{% bs_icon 'trash' %}</span>
        </button>
    </form>
</div>
```

¿Qué destacamos?

- El método del formulario es POST
- Creamos un botón con un icono
- Llamamos a una función javascript, que preguntará si confirmamos esa eliminación

Delete-Eliminar

La función javascript, sería de la siguiente forma:

```
function eliminar() {
    var x = confirm("¿Eliminar Libro ?");
    if (x)
      return true;
    else
      return false;
}
```

Recuerdo que debemos Incluir el javascript de la siguiente forma:

```
{% block cabecera_extra %}
    {% load static %}
    <script type="text/javascript" src="{% static 'javascript/principal.js' %}" ></script>
{% endblock %}
```

Delete-Eliminar

RETO: ¿Cómo haríamos el delete con un modal de Bootstrap?

Mensajes

Por último, para mejorar la interfaz de usuario, es importante notificar siempre, cuál es la operación que hemos realizado.

Para ello usaremos los mensajes, y debemos incluirlos siempre que hagamos una operación de Create, Update y Delete.

Mensajes

1º Importamos la librería

```
from django.contrib import messages
```

2º Creamos el mensaje antes de realizar el redirect

```
messages.success(request, 'Se ha creado el libro'+formulario.cleaned_data.get('nombre')+" correctamente")
```

3º Incluimos el código en las plantillas para mostrar la alerta

```
{% if messages %}
    {% for message in messages %}
        <div class="alert alert-{{ message.tags }} alert-dismissible fade show" role="alert">
            {{ message }}
            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
        </div>
    {% endfor %}
{% endif %}
```

Otra opción es usar bootstrap:

{% bootstrap_messages %}