**Ejemplo de integración de Django con React**

https://www.youtube.com/watch?v=9U5F\_BtPsLU

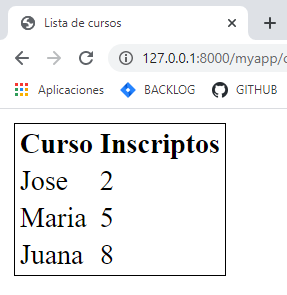
**FORMULARIOS** (Definimos y generamos un formulario en Django):

1.Los formularios nos permiten no tener que escribir el código HTML. Es decir, validan que los datos ingresados sean pertinentes.

2. Partimos del siguiente código:



Y llegamos al siguientes resultado a partir de <http://127.0.0.1:8000/myapp/cursos>, (en donde gracias a la función **cursos** se hace un select general de la tabla cursos):



3. Implementamos un **Formulario** para insertar cursos. Creamos **myapp/forms.py**:

from django import forms

class FormularioCurso(forms.Form):

    nombre = forms.CharField(label="Nombre", max\_length=128)

    inscriptos = forms.IntegerField(label="Inscriptos")

el campo **nombre**  de tipo **forms.CharField** (texto de una línea, máximo 128 caracteres). Y el campo **inscriptos** de tipo **forms.IntegerField** (un entero, con un titulo llamado “Inscriptos”). El texto ingresado se convertirá a entero.

4. **views.py:** importamos el formulario al inicio del archivo y definimos una vista en la que instanciamos dicho formulario.

Una misma **definición** puede servir para crear multiples instancias de **formularios** en distintas vistas.

1. Importamos el formulario:

from . import forms

1. Definimos la vista:

def nuevo\_curso(request):

    form = forms.FormularioCurso()

    ctx = {"form": form}

    return render(request, "myapp/nuevo\_curso.html", ctx)

5. Creamos **nuevo\_curso.html**:

<h1>Nuevo curso</h1>

<form method="post"> <!--Codigo HTML tradicional para generar un formulario-->

                     <!--El metodo HTTP POST envia la peticion al servidor-->

{% csrf\_token %}     <!--Es una variable que Django agrega automaticamente al contexto

                       de nuestra plantilla. Y que genera un codigo que protege el

                       formulario de ciertas vulnerabilidades

                       https://es.wikipedia.org/wiki/Cross-site\_request\_forgery-->

{{ form.as\_p}}      <!--genera el codigo HTML para los campos que definimos

                         en forms.FormularioCurso-->

                         <!--Se puede usar solamente {{form}}, para generar el formulario

                        en una misma linea. El agregado as\_p indica que se deben usar

                    etiquetas <p> para cada campo, lo que hara que se muestren uno por linea-->

<input type="submit" value="Añadir curso">

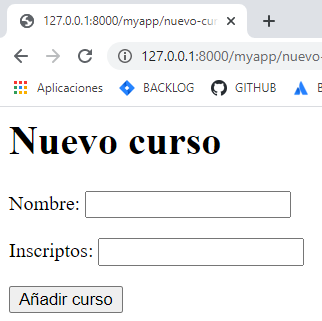
</form>

6. Asociamos la vista con la URL, **myapp/urls.py**:

path("nuevo-curso", views.nuevo\_curso, name="nuevo\_curso")

7. Iniciamos el server (python manage.py runserver).

8. Nos dirigimos a la URL: <http://127.0.0.1:8000/myapp/nuevo-curso> (por ahora mostramos el formulario preparado)



9. Codigo resultante:



**Notas:** La librería que permite el envio de datos es **requests.**

**ENVIO Y VALIDACION** (de los datos del formulario)

En cada petición que el browser realiza a nuestra app de Django se incluye un metodo **HTTP**.

**GET:** es el metodo mas utilizado, indica que se quiere obtener el contenido de una pagina en particular.

**POST:** Enviar datos al servidor. Se usa para **formularios.**

1.Partimos del código anterior (punto 9).

Una **vista** utiliza el **GET** cuando esta siendo invocada (debe mostrar el formulario).

Y utiliza el **POST** cuando recibe y valida los datos.

2. Hacemos las modificaciones pertinentes a **nuevo\_curso** (Pag 4, ver marca en amarillo):

def nuevo\_curso(request):

    if request.method == "POST":

        form = forms.FormularioCurso(request.POST) # cuando el metodo es POST deben recibirse los

        if form.is\_valid():                       #datos del formulario y validarse

            return HttpResponse("¡Curso creado correctamente!")

    else:

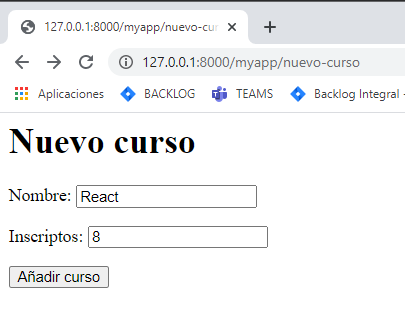
        form = forms.FormularioCurso() #para cualquier otro metodo: PUT, DELETE, HEAD, se mostrara

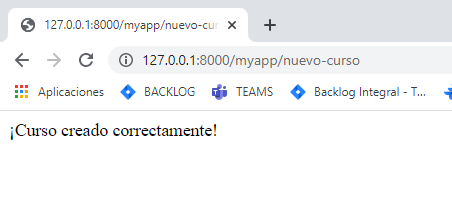
                                       #el formulario

    ctx = {"form": form}

    return render(request, "myapp/nuevo\_curso.html", ctx)

3. <http://127.0.0.1:8000/myapp/nuevo-curso>: por ahora no impactamos con la base, simplemente agregamos la inteligencia a la función para que discrimine entre POST y otros métodos. Y también agregamos una respuesta para el usuario.

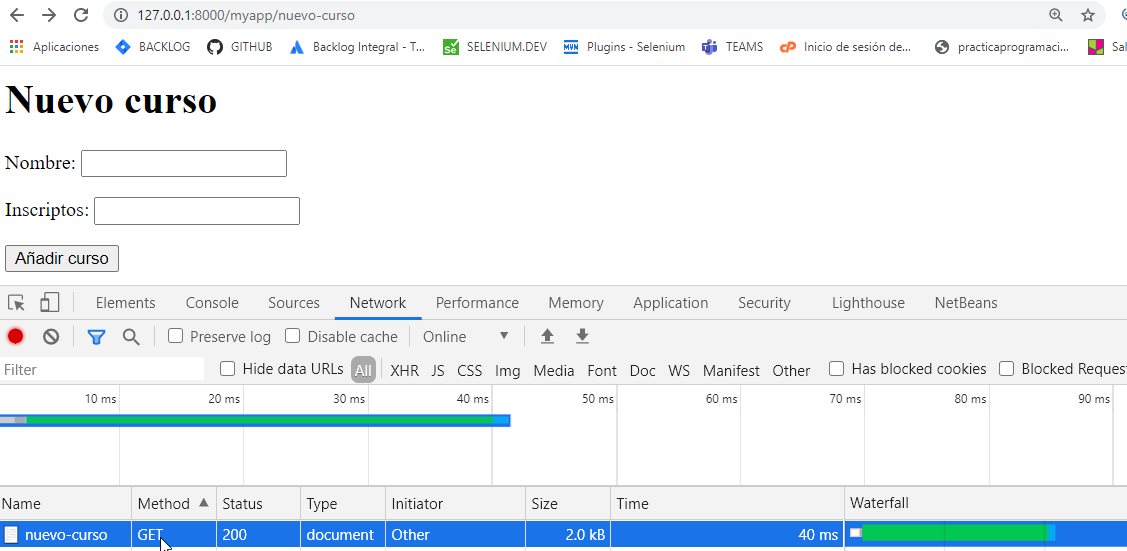




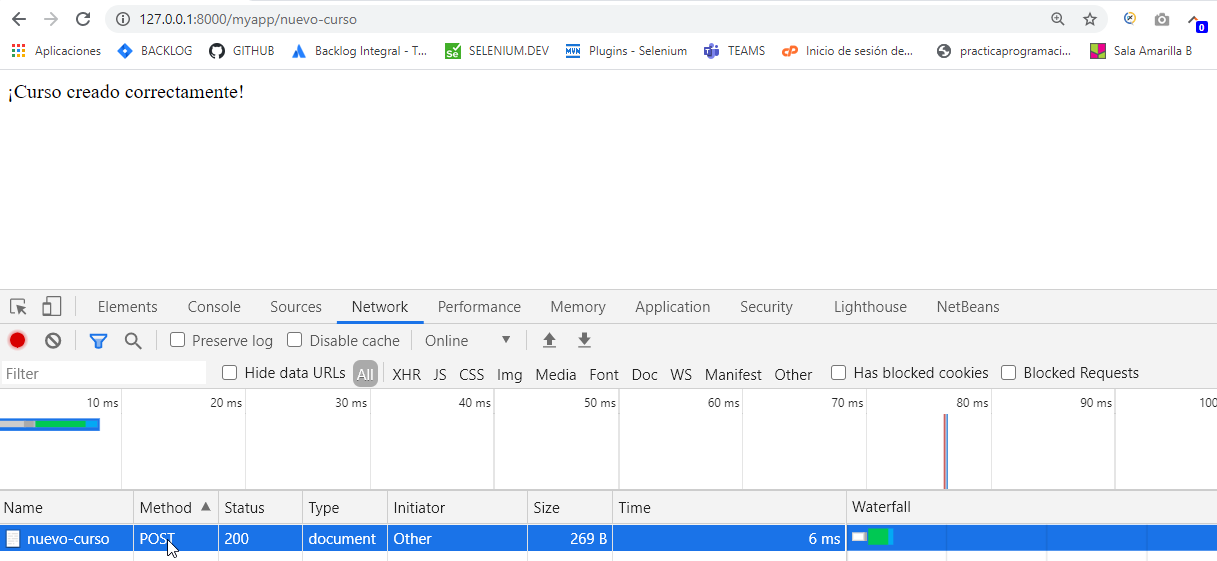
**Observamos como funciona el envio de datos.**

4. Hacemos un F12 en http://127.0.0.1:8000/myapp/nuevo-curso y vemos como interviene en la 1er pantalla el metodo **GET** y en la segunda pantalla el metodo **POST:**

1ra: El navegador realiza una petición GET.



2da: Al oprimir “Añadir Curso” el navegador realizar una petición POST.



*Seguimos con:*

5. **Validaciones.**

Luego leer las paginas 4 y 5 de validaciones

6. Creamos la siguiente tabla (aunque podemos usar también la de cursos):

create table cursos1 (

id integer primary key autoincrement,

nombre\_curso text,

inscriptos integer

)

7. Considerando las **validaciones** del punto modificamos **nuevo\_curso** (agrega un curso a la BBDD):

def nuevo\_curso(request):

    if request.method == "POST":

        form = forms.FormularioCurso(request.POST)

        if form.is\_valid():

            conn = sqlite3.connect("db.sqlite3")

            cursor = conn.cursor()

            cursor.execute(

                "INSERT INTO cursos1 (nombre\_curso, inscriptos) VALUES (?, ?);",

                (form.cleaned\_data["nombre"], form.cleaned\_data["inscriptos"])

            )

            conn.commit()

            conn.close()

            return HttpResponse("¡Curso creado correctamente!")

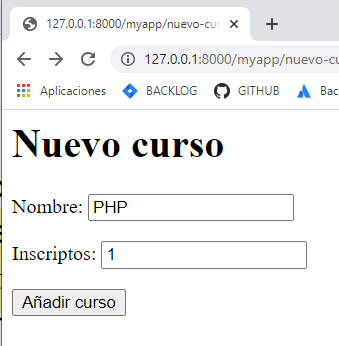
    else:

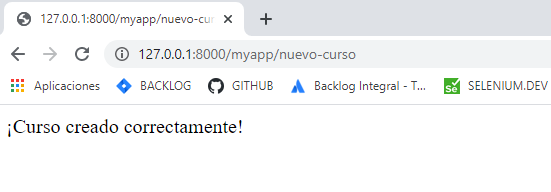
        form = forms.FormularioCurso()

    ctx = {"form" : form}

    return render(request, "myapp/nuevo\_curso.html", ctx)

8. <http://127.0.1:8000/myapp/nuevo-curso>





**REENVIAMOS LA LISTA DE CURSOS de la tabla CURSOS1**

9. Cuando el formulario se haya validado correctamente y el curso creado, se reenvía al usuario la lista de cursos. Para eso, en lugar de retornar **HttpResponse**, vamos a usar **HttpResponseRedirect**. Vamos a necesitar también la función **django.urls.reverse(),** cuya utilidad es la de generar direcciones de URL desde las vistas (al igual que usábamos la etiqueta {% url … %} en las plantillas).

1. Agregamos las siguientes líneas al inicio de **views.py**:

from django.http import HttpResponseRedirect

from django.urls import reverse

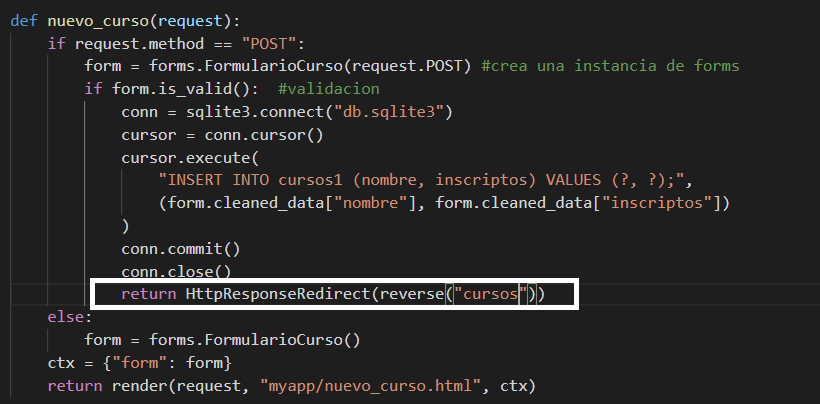
1. Despues en lugar de

Return HttpResponse(“¡Curso creado correctamente!”)

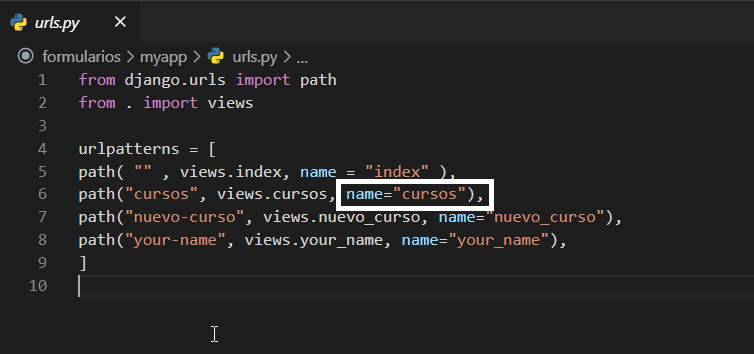
ponemos

return HttpResponseRedirect(reverse("cursos"))

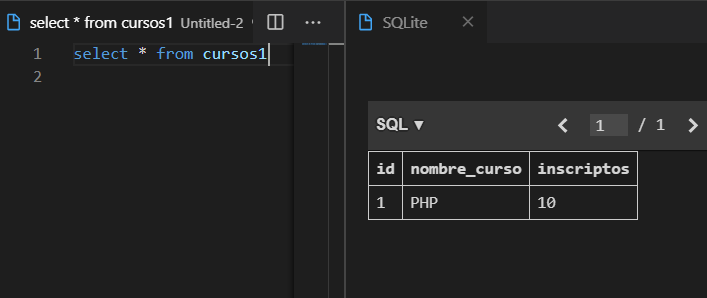
1. De forma tal que nos quedaría:



10. Esa cadena **cursos** es el nombre que habíamos configurado para la dirección de URL en **urls.py**:



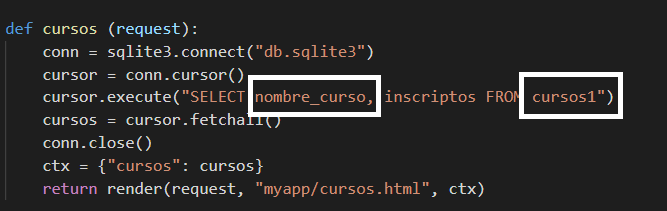
11. Le hacemos un select a la tabla cursos o cursos1 para ver lo que tenemos:



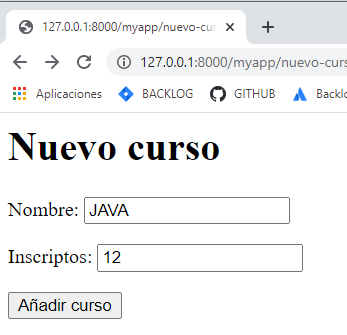
Entonces la idea es que cuando el usuario se dirija a <http://127.0.0.1:8000/myapp/nuevo-curso> pueda cargar el curso en la tabla cursos1 y luego cuando aprete el botón **AÑADIR CURSO** que inserte el registro y que lo redireccione a <http://127.0.0.1:8000/myapp/cursos> para mostrar los cursos existentes.

**Nota:**

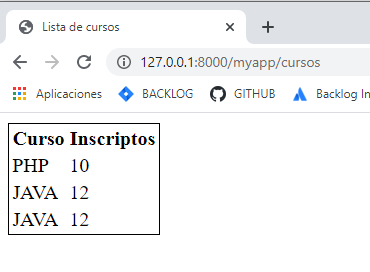
Tener en cuenta la pequeña modificación que hay que hacer en la vista **cursos** para que realice el select sobre la tabla correcta



12. Entonces vamos a <http://127.0.0.1:8000/myapp/nuevo-curso> , ingresamos un curso y damos **añadir curso:**

****

Y obtenemos:



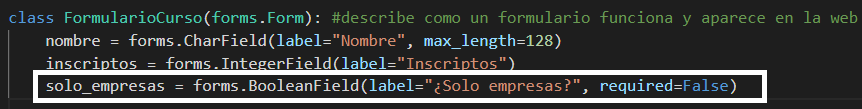
13. Codigo resultante:



**OTROS CAMPOS** (se muestra como agregar un campo boolean al form de Django)

1.**Booleano:** representado por la clase **django.forms.BooleanField**. Por defecto el código HTML genera una casilla de verificación. El valor recibido en **form.cleaned\_data** es un ­**bool**.

2. Tomamos el código previo como base. Y hacemos un agregado en **forms.py**:



Donde **required=False** es necesario para permitir que el campo este vacio. Es decir como es una casilla de verificación su valor debe ser Falso.

Para que **impacte en la base** debemos hacer algunas modificaciones:

**cursos.html** agregar:

  <th>Solo Empresas</th>

y

 {% for nombre, inscriptos, solo\_empresas in cursos %}

            <tr>

                <td> {{ nombre }} </td>

                <td> {{ inscriptos }} </td>

                <td>{{ solo\_empresas }}</td>

            </tr>

        {% endfor %}

**views.py** Funcion **cursos** actualizamos en el select:

cursor.execute("SELECT nombre\_curso, inscriptos, solo\_empresas FROM cursos1")

**nuevo\_curso** actualizamos en el execute():

cursor.execute(

                "INSERT INTO cursos1 (nombre\_curso, inscriptos, solo\_empresas) VALUES (?, ?, ?);",

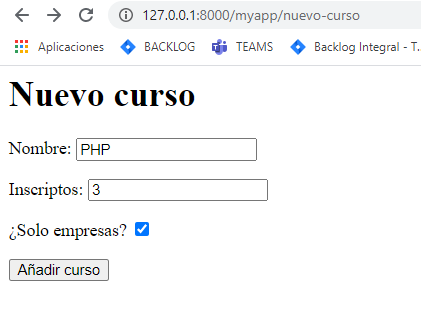
                (form.cleaned\_data["nombre"], form.cleaned\_data["inscriptos"], form.cleaned\_data["solo\_empresas"])

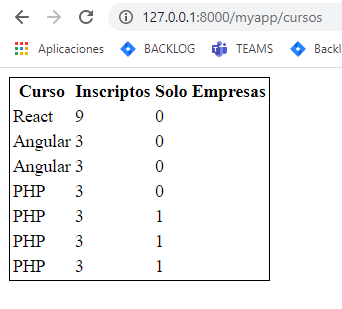
            )

Tabla **cursos1**

alter table cursos1 ADD COLUMN solo\_empresas boolean default false

3. Chequeamos <http://127.0.0.1:8000/myapp/nuevo-curso>:





**Codigo Resultante:**



**LISTA DE OPCIONES**

1.Clase **django.forms.ChoiceField**. Por defecto el código HTML generado es una etiqueta **select**. Cada **opción** de esta **lista** está representado por una **tupla**. El primer elemento de la tupla indica el valor que recibirá el formulario por cada elemento seleccionado, mientras que el segundo es el texto que se le mostrara al usuario.

2. Hacemos una modificación en la clase **FormularioCurso(forms.Form)** en **forms.py**:

class FormularioCurso(forms.Form): #describe como un formulario funciona y aparece en la web

    nombre = forms.CharField(label="Nombre", max\_length=128)

    inscriptos = forms.IntegerField(label="Inscriptos")

    solo\_empresas = forms.BooleanField(label="¿Solo empresas?", required=False) #va False porque es una casilla de verificacion, no debe aparecer tildada

    TURNOS = (

        (1, "Mañana"),

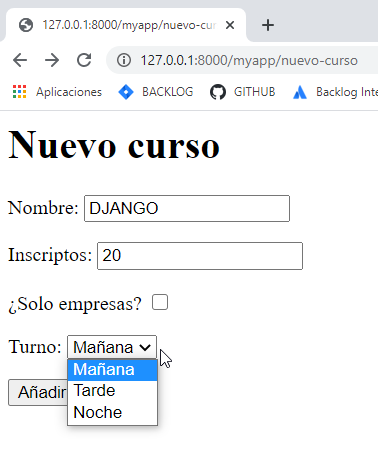
        (2, "Tarde"),

        (3, "Noche")

    )

    turno = forms.ChoiceField(label="Turno", choices=TURNOS)

3. Chequeamos <http://127.0.0.1:8000/myapp/nuevo-curso>:



3. Si el usuario seleccióna “Mañana”, el **form.cleaned\_data[“turno”]** será 1 (puesto que es el valor que definimos en la tupla pasada al argumento **choices**). Si seleccina “Tarde”, será 2. Y para “Noche” será 3.

1. Creamos una tabla **cursosDetalle:**

-- SQLite

create table cursosDetalle (

id integer primary key autoincrement,

nombre\_curso text,

inscriptos integer,

soloEmpresas boolean,

turno text

)

1. Hacemos una pequeña modificación en **views.py – nuevo\_curso**:

Nota: ver el cambio del campo solo\_empresas a soloEmpresas

 cursor.execute(

                "INSERT INTO cursosDetalle (nombre\_curso, inscriptos, soloEmpresas, turno) VALUES (?, ?, ?, ?);",

                (form.cleaned\_data["nombre"], form.cleaned\_data["inscriptos"], form.cleaned\_data["soloEmpresas"], form.cleaned\_data["turno"])

            )

1. Hacemos una modificación en **views.py – cursos:**

cursor.execute("SELECT nombre\_curso, inscriptos, soloEmpresas, turno FROM cursosDetalle")

1. Modificamos **cursos.html** para que cuando sea llamado, considere el campo “turno”:

 <thead>

            <tr>

                <th> Curso </th>

                <th> Inscriptos </th>

                <th> Es Empresa? </th>

                <th> Turno </th>

            </tr>

        </thead>

        {% for nombre, inscriptos, soloEmpresas, turno in cursos %}

            <tr>

                <td> {{ nombre }} </td>

                <td> {{ inscriptos }} </td>

                <td> {{ soloEmpresas }} </td>

                <td> {{ turno }} </td>

4. Chequeemos que **forms.py** tenga los siguientes campos:

class FormularioCurso(forms.Form):

    nombre = forms.CharField(label="Nombre", max\_length=128)

    inscriptos = forms.IntegerField(label="Inscriptos")

    soloEmpresas = forms.BooleanField(label="¿Solo empresas?", required=False)

    TURNOS=(

        (1, "Mañana"),

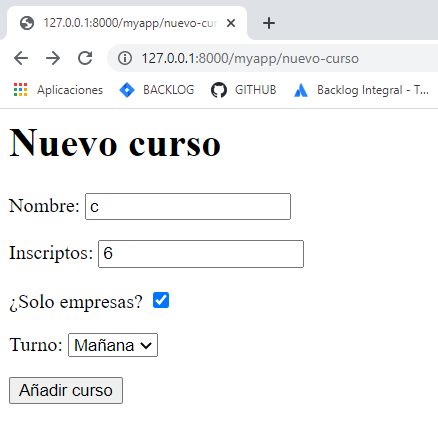
        (2, "Tarde"),

        (3, "Noche")

    )

    turno = forms.ChoiceField(label="Turno", choices=TURNOS)

5. Chequeamos:





5. Codigo Resultante:



**FECHA** (pag 9)

1.Partimos del código previo. Clase **django.forms.DateField**. Por defecto el código HTML generado es el mismo que para django.forms.CharField (caja de texto). La diferencia es que el primero validara que la fecha se haya ingresado en el formato correcto, y el tipo de dato del valor del campo en **form.cleaned\_data** será **datetime.date**.

2. Agregamos lo siguiente a nuestra clase **FormularioCurso**:

fecha\_inicio = forms.DateField(

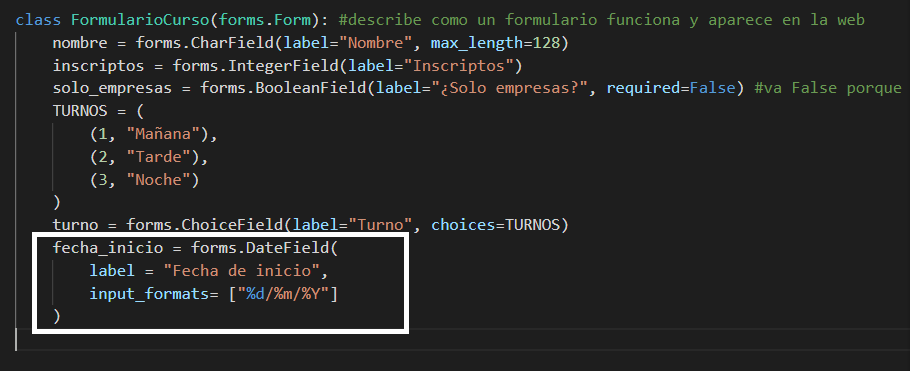
        label = "Fecha de inicio",

        input\_formats= ["%d/%m/%Y"]

    )

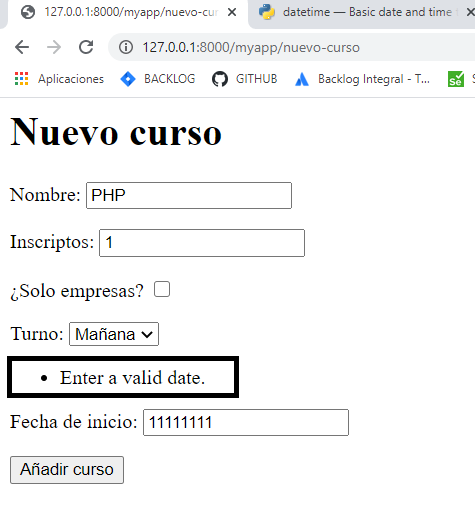
De forma tal que nos queda:

**forms.py**



Indicamos que el formato debe ser %d/%m/%Y. Ver <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes> para el significado de cada uno y los posibles formatos.

3. Probamos, como al ingresar cualquier numero nos valida el campo fecha:



Ingresamos la fecha de hoy y vemos como la toma: 01/03/2021

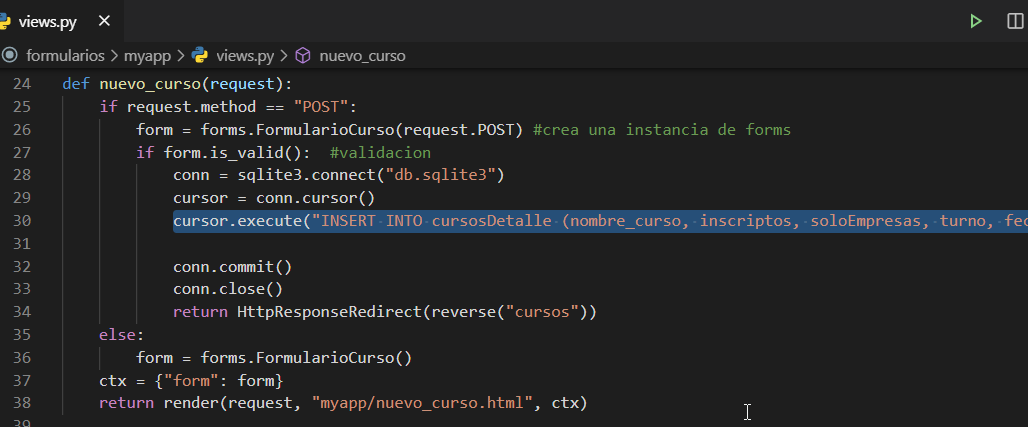
4. Cargamos la fecha en la Base de Datos desde Django:

1. **BBDD:** Tabla cursosDetalle. Agregamos un campo fecha al final.

alter table cursosDetalle ADD COLUMN fecha TEXT

1. **views.py: ­nuevo\_curso**

cursor.execute("INSERT INTO cursosDetalle (nombre\_curso, inscriptos, soloEmpresas, turno, fecha) VALUES (?, ?, ?, ?, ?);", (form.cleaned\_data["nombre"], form.cleaned\_data["inscriptos"], form.cleaned\_data["solo\_empresas"], form.cleaned\_data["turno"], form.cleaned\_data["fecha\_inicio"]) )



El nombre del parámetro de **form.cleaned\_data** tiene que ser igual a la variable que pusimos en forms.py. Nota al margen, recordemos que lo que hace **form.cleaned\_data** es validar y normalizar el ingreso de un valor fecha.

**cursos:** agregamos el campo fecha.

def cursos (request):

    conn = sqlite3.connect("db.sqlite3")

    cursor = conn.cursor()

    cursor.execute("SELECT nombre\_curso, inscriptos, soloEmpresas, turno, fecha FROM cursosDetalle")

    cursos = cursor.fetchall()

    conn.close()

    ctx = {"cursos": cursos}

    return render(request, "myapp/cursos.html", ctx)

5. Modificamos **cursos.hmlt** para que se pueda ver la fecha:

          <th>Fecha Inicio</th>

Y

  {% for nombre, inscriptos, soloEmpresas, turno, fecha\_inicio in cursos %}

            <tr>

                <td> {{ nombre }} </td>

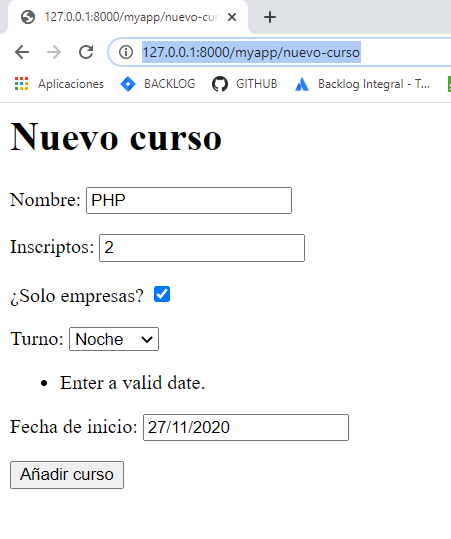
                <td> {{ inscriptos }} </td>

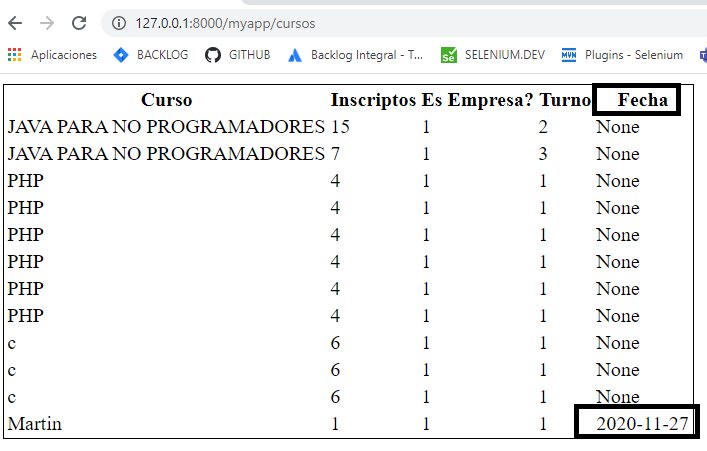
                <td>{{ soloEmpresas }}</td>

                <td>{{ turno }}</td>

                <td>{{ fecha\_inicio }}</td>

6. <http://127.0.0.1:8000/myapp/nuevo-curso>:





6. Codigo Resultante:



7. Para que en lugar de que salga un “1” o “0” en soloEmpresas incluimos la siguiente inteligencia en **cursos.html** para que coloque “SI” o “NO”.

**cursos.html**

{% for nombre, inscriptos, soloEmpresas, turno, fecha\_inicio in cursos %}

            <tr>

                <td> {{ nombre }} </td>

                <td> {{ inscriptos }} </td>

        <!--  <td> {{ soloEmpresas }} </td>  -->

                <td>{% if soloEmpresas  %}

                    Si.

                 {% else %}

                    No.

                 {% endif %}</td>

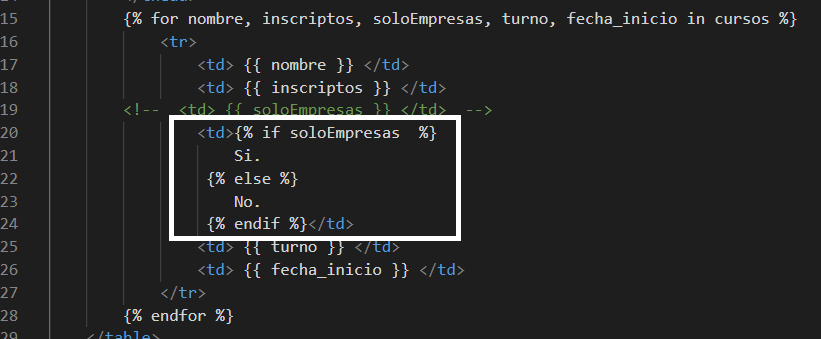
                <td> {{ turno }} </td>

                <td> {{ fecha\_inicio }} </td>

            </tr>

        {% endfor %}

Se muestran las modificaciones realizadas:



Y si queremos poner una inteligencia también para los turnos “Mañana, tarde o noche” en lugar de “1, 2, 3” debemos hacer el siguiente reemplazo:

<td>

                     {% ifequal turno '1' %}

                        Turno Mañana

                        {% endifequal %}

                     {% ifequal turno '2'  %}

                        Turno Tarde

                        {% endifequal %}

                     {% ifequal turno '3'  %}

                        Turno Noche

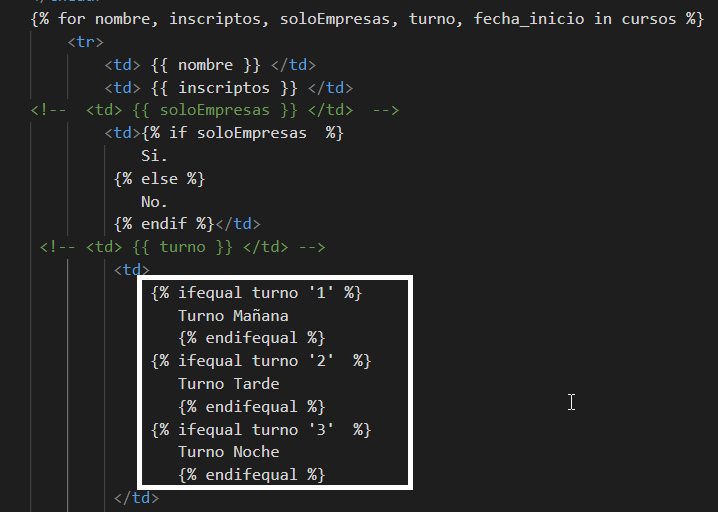
                        {% endifequal %}

                 </td>

En lugar de

<td> {{ turno }} </td>

Tal como se muestra a continuación:



**Checkpoint Codigo:**

****

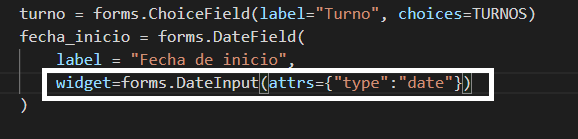
**CALENDARIO** (para seleccionar la fecha, pag 9)

1.**forms.py** (tomamos como base el código previo):

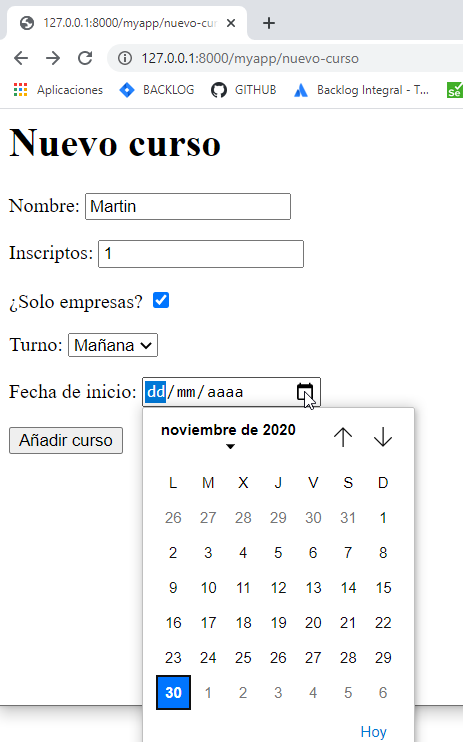
Hacemos una pequeña modificación en **fecha\_inicio**:

widget=forms.DateInput(attrs={"type":"date"})

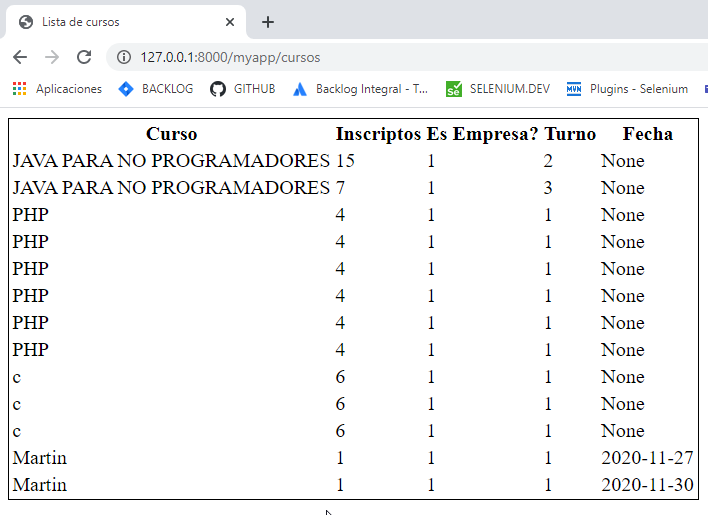
de forma tal que nos queda:



2. Observamos la diferencia:



3. Resultado:



4. Lista completa de campos provistos por Django:

<https://docs.djangoproject.com/en/stable/ref/forms/fields/#built-in-field-classes>

**LABORATORIO**

Ejercicio\_1

1.Partimos del siguiente código:



2. Iniciamos el servidor.

3. Abrimos la base de datos **db.sqlite3** en donde creamos la tabla peliculasEstreno:

create table peliculasEstreno (

id integer primary key autoincrement,

nombrePeli text,

fechaEstreno text,

mayoresDe integer,

preventaOnline boolean

)

4. **forms.py**:

from django import forms

class FormularioPelicula(forms.Form):

    nombrePeli = forms.CharField(label="Nombre", max\_length=128)

    fechaEstreno = forms.DateField(

        label="Fecha de estreno",

        widget=forms.DateInput(attrs={"type": "date"})

    )

    mayoresDe = forms.IntegerField(label="Para mayores de")

    preventaOnline = forms.BooleanField(

        label="¿Preventa online?",

        required=False

    )

5. Agregamos los siguientes import en **views.py**: Si bien el ejercicio no pide ingresar en la BBDD, nosotros lo agregamos para practicar.

import requests, sqlite3

from django.http import HttpResponse

from . import forms

from django.shortcuts import render

ingresar una película nueva:

def nueva\_pelicula(request):

    if request.method == "POST":

        form = forms.FormularioPelicula(request.POST) #crea una instancia de forms

        if form.is\_valid():  #validacion

            conn = sqlite3.connect("db.sqlite3")

            cursor = conn.cursor()

            cursor.execute("INSERT INTO peliculasEstreno (nombrePeli, fechaEstreno, mayoresDe, preventaOnline) VALUES (?, ?, ?, ?);", (form.cleaned\_data["nombrePeli"], form.cleaned\_data["fechaEstreno"], form.cleaned\_data["mayoresDe"], form.cleaned\_data["preventaOnline"]) )

            conn.commit()

            conn.close()

            ctx= {"pelicula": form.cleaned\_data} #se pasa como variable de contexto el formulario validado

            return render(request, "myapp/peliculaInsertada.html", ctx)

    else:

        form = forms.FormularioPelicula()

    ctx = {"form": form}

    return render(request, "myapp/nueva\_pelicula.html", ctx)

6. **urls.py** (urlpatterns) agregamos:

 path("nueva-pelicula", views.nueva\_pelicula, name="nueva\_pelicula")

7. **Templates**: Creamos una carpeta templates/myapp en donde colocamos las siguientes plantillas.

**peliculaInsertada.html:** Muestra los datos ingresados una vez que el formulario ha sido llenado y validado.

<html>

  <p>Se ha ingresado correctamente la siguiente película:</p>

  <p>Nombre: {{ pelicula.nombrePeli }}.</p>

  <p>Fecha de estreno: {{ pelicula.fechaEstreno }}.</p>

  <p>Para mayores de {{ pelicula.mayoresDe }} años.</p>

  <p>

    Preventa online:

    {% if pelicula.preventaOnline %}

      Sí.

    {% else %}

      No.

    {% endif %}

  </p>

</html>

**nueva\_pelicula:** para mostrar el formulario.

<html>

    <title>Nueva película</title>

    <h1>Nueva Pelicula</h1>

    <form method="post">

        {% csrf\_token %}

        {{ form.as\_p }}

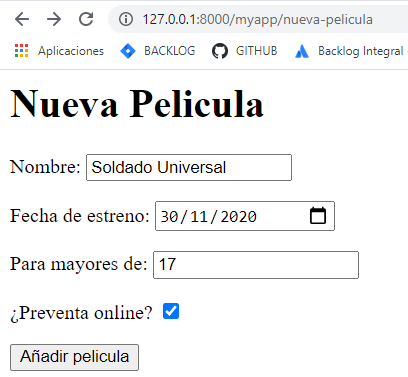
    <input type="submit" value="Añadir pelicula">

</form>

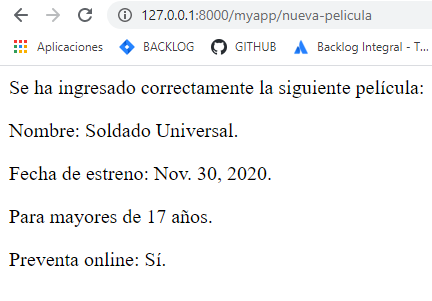
</html>

8. Reiniciar el servidor python manage.py runserver

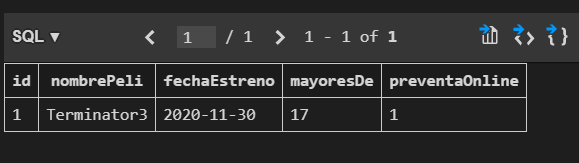
9. <http://127.0.0.1:8000/myapp/nueva-pelicula>:



10. Resultado:



11. Tabla **peliculasEstreno**



12. Codigo Resultante:



**PROYECTO INTEGRADOR:**

1.Partimos del siguiente proyecto:

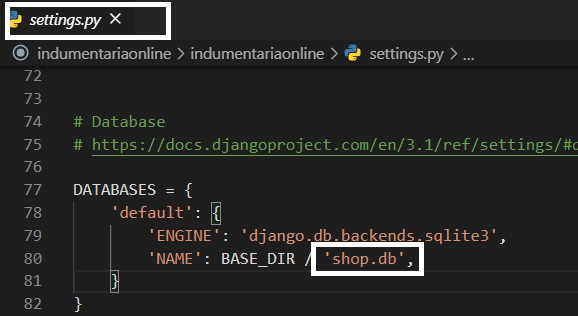


2. Iniciamos el servidor: python manage.py runserver

3. Partimos de la siguiente url <http://127.0.0.1:8000/shop/>:

4. En el nuevo integrador vamos a definir un formulario para que los administradores del sitio puedan cargar el catalogo, y luego mostrarlo al inicio. El producto será “Remeras”.

5. Creamos la base **shop.db** cambiando el nombre de la base de datos que viene con Django por Default:



Luego hay que seleccionar “créate database” desde shop.db.

6. Creamos la tabla **remeras**:

create table remeras (

marca TEXT,

talle NUMERIC,

color TEXT,

lisa BOOLEAN,

genero NUMERIC

)

7. Creamos el formulario **FormularioRemera** en **forms.py**:

from django import forms

class FormularioRemera(forms.Form):

    marca = forms.CharField(label="Marca", max\_length=128)

    TALLES = (            #linkeo de los numeros con el valor correspondiente

        (1, "XS"),

        (2, "S"),

        (3, "M"),

        (4, "L"),

        (5, "XL"),

        (6, "XXL")

    )

    talle = forms.ChoiceField(label="Talle", choices=TALLES)

    color = forms.CharField(label="Color", max\_length=128)

    lisa = forms.BooleanField(label="Lisa", required=False)

    GENEROS = (

        (1, "Hombre"),

        (2, "Mujer"),

        (3, "Unisex")

    )

    genero = forms.ChoiceField(label="Género", choices=GENEROS)

8. **urls.py**:

    path("nueva-remera", views.nueva\_remera, name="nueva\_remera")

9. **views.py**:

Importamos

import sqlite3

from django.http import HttpResponseRedirect

from django.shortcuts import render

from django.urls import reverse

from .forms import FormularioRemera

**nueva\_remera:**

def nueva\_remera(request):

   if request.method == "POST":

      form = forms.FormularioRemera(request.POST)

      if form.is\_valid():

         conn = sqlite3.connect("shop.db")

         cursor = conn.cursor()

          cursor.execute(

                "INSERT INTO remeras VALUES (?, ?, ?, ?, ?)",

                (

                    form.cleaned\_data["marca"],

                    form.cleaned\_data["talle"],

                    form.cleaned\_data["color"],

                    form.cleaned\_data["lisa"],

                    form.cleaned\_data["genero"]

                )

            )

         conn.commit()

         conn.close()

         return HttpResponseRedirect(reverse("index")) #aqui se pone el nombre de la url destino, name="index"

   else:

        form = FormularioRemera()

   ctx = {"form": form}

   return render(request, "shop/nueva\_remera.html", ctx)

**index:**

def index(request):

   conn = sqlite3.connect("shop.db")

   cursor = conn.cursor()

   cursor.execute("SELECT \* FROM remeras")

   remeras = cursor.fetchall()

   conn.close()

   ctx = {"remeras": remeras}

   return render(request, "shop/index.html", ctx)

10. Plantillas:

**Index.html:**

{% extends "shop/base.html" %}

{% block title %}Pagina Inicial {%endblock %}

{% block content %}

<h1>¡Bienvenido a nuestro sitio!</h1>

<h3>Catalogo</h3>

<table>

    <thead>

        <tr>

            <th>Marca</th>

            <th>Talle</th>

            <th>Color</th>

            <th>Lisa</th>

            <th>Genero</th>

        </tr>

    </thead>

    {% for marca, talle, color, lisa, genero in remeras %}

        <tr>

            <td>{{ marca }}</td>

            <td>{{ talle }}</td>

            <td>{{ color }}</td>

            <td>{{ lisa }}</td>

            <td>{{ genero }}</td>

        </tr>

    {% endfor %}

</table>

<p>

    <a href="{% url 'contacto' %}">Ir a contacto</a>

</p>

{%endblock%}

**nueva\_remera.html**

{% extends "shop/base.html" %}

{% block content %}

<h1>Nueva remera</h1>

<form method="post">

    {% csrf\_token %} <!--variable que Django agrega automaticamente al contexto

                         de nuestra plantilla. Y que protege al formulario de ciertas

                        vulnerabilidades-->

    {{ form.as\_p }}

    <input type="submit" value="Guardar">

</form>

{% endblock %}

11. Migramos python manage.py migrate.

E iniciamos el servidor: python manage.py runserver

12. <http://127.0.0.1:8000/shop/nueva-remera>



12. Resultado:



13. Codigo Resultante:



14. Diferentes formas de mostrar un formulario en el template:

**nueva\_remera.html**

cambiamos esto

<form method="post">

    {% csrf\_token %} <!--variable que Django agrega automaticamente al contexto

                         de nuestra plantilla. Y que protege al formulario de ciertas

                        vulnerabilidades-->

    {{ form.as\_p }}

    <input type="submit" value="Guardar">

</form>

Por esto:

<form method="post">

    {% csrf\_token %} <!--variable que Django agrega automaticamente al contexto

                         de nuestra plantilla. Y que protege al formulario de ciertas

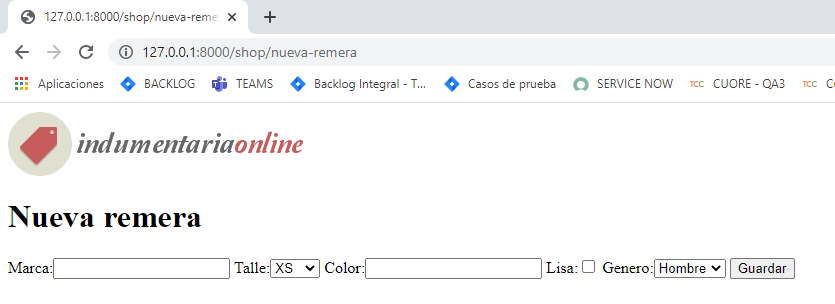
                        vulnerabilidades-->

    {{ form }}

    <input type="submit" value="Guardar">

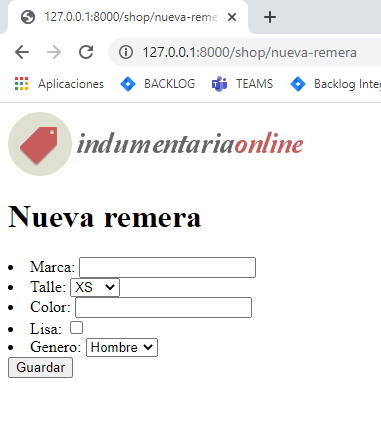
</form>

Y vemos el formulario en forma apaisada:



También con la misma lógica lo podemos mostrar como una lista desordenada **ul**:

    {{ form.as\_ul }}

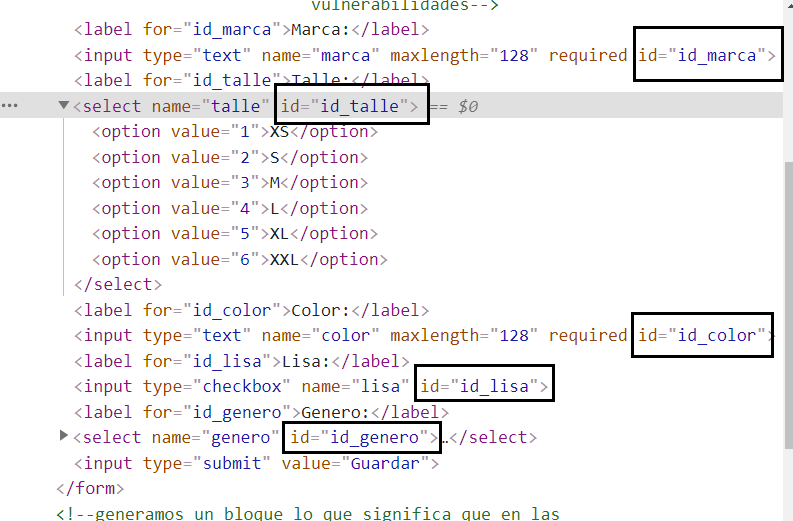


Y vemos el formulario como una lista (F12).

15. Manejo del **ID**. La API Forms determina automáticamente un id al atributo del formulario el cual se volcara al html con la estructura **id\_formFieldName**

a.

Ejemplo: A continuación se muestran en el F12 los id generados automáticamente:



Con lo cual con esto ya podemos agregar estilos de css e indicar al programador de css que tiene disponible el **id** para tal efecto.

b. Imprimimos el formulario:

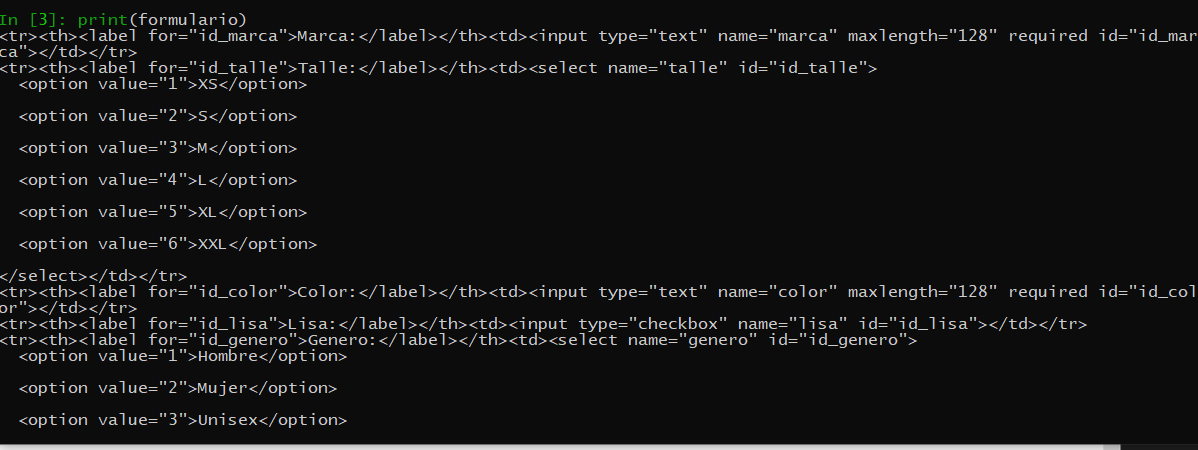
python manage.py Shell

In [1]: from shop.forms import FormularioRemera

In [2]: formulario = FormularioRemera()

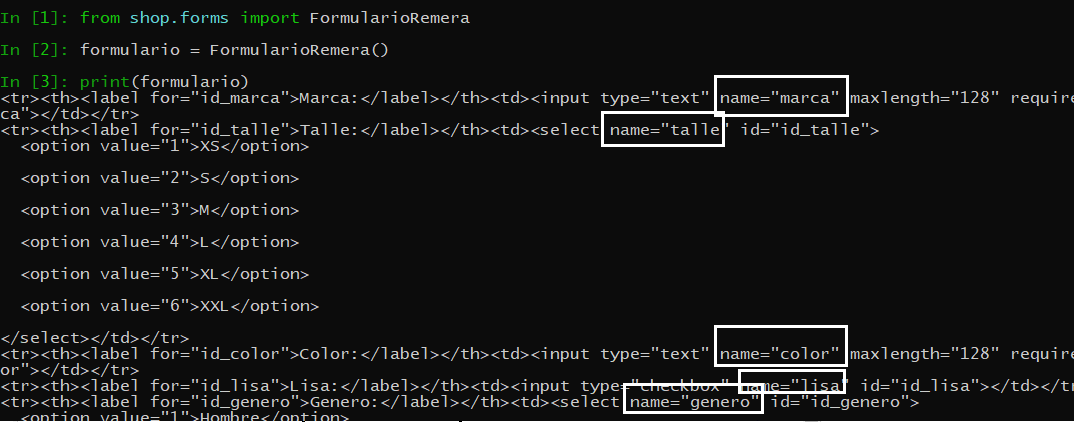
In [3]: print(formulario)

Notese como lo formatea como una tabla **por defecto**:



Tambien crea etiquetas **label** de forma automática. Tambien crea de forma automática los **input type.** Por defecto los campos de tipo texto son **requeridos**.

1. Para enviar ciertos datos cuando se instancia el formulario apuntamos a la propiedad **name** de cada campo:



Entonces por ejemplo si queremos popular de datos via un **diccionario** hacemos:

formulario=FormularioRemera({'marca': 'Adidas', 'talle': '1', 'color': 'blue', 'lisa': '1', 'genero': '1'})

Lo validamos:

In [8]: formulario.is\_valid()

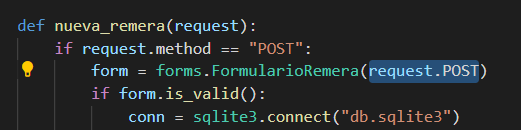
Y luego con la **propiedad** cleaned\_data obtenemos el diccionario que le habíamos pasado:

In [9]: formulario.cleaned\_data

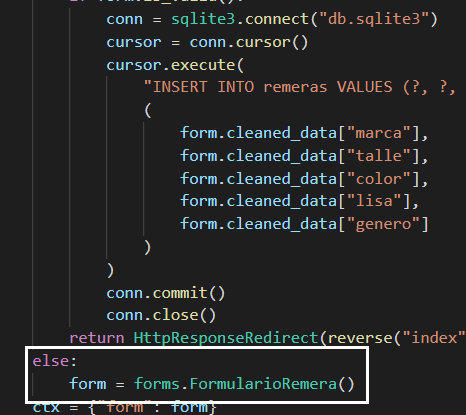
Out[9]: {'marca': 'Adidas', 'talle': '1', 'color': 'blue', 'lisa': True, 'genero': '1'}

Dicho esto cabe aclarar algunas cosas:

1. El **request.POST** que incluimos en la función **nueva\_remera** cuando instanciamos el formulario es para tener disponible los datos que ingreso el usuario:



1. El **else** de la función sin embargo pone a disposición el formulario para que lo vea el usuario **ANTES** de haber ingresado los datos. Por eso no lleva **request.POST** ya que usa el metodo **GET**.



1. Cabe aclarar que en el template el formulario de Django **no genera** la etiqueta **form** ni la etiqueta **table**. Por eso la tenemos que generar nosotros:

Entonces modificamos el template **nueva\_remera.html**:

{% extends "shop/base.html" %}

{% block content %}

<h1>Nueva remera</h1>

<form method="post">

    {% csrf\_token %}

    <!--variable que Django agrega automaticamente al contexto

                         de nuestra plantilla. Y que protege al formulario de ciertas

                        vulnerabilidades-->

    <table>

        {{ form.as\_table }}

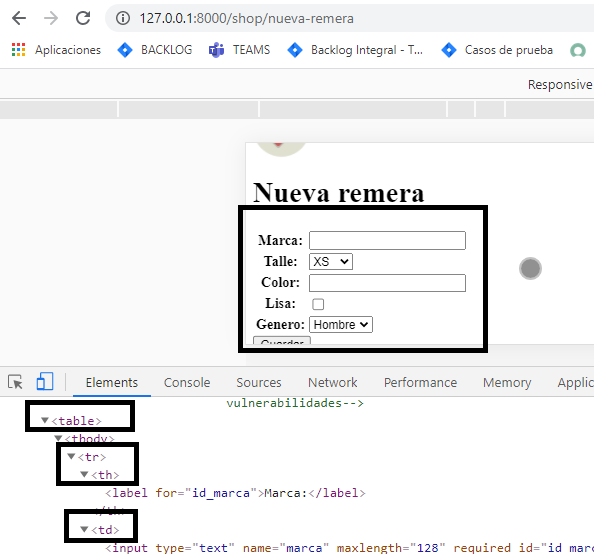
    </table>

    <input type="submit" value="Guardar">

</form>

{% endblock %}

Y viéndolo desde el F12 vemos como nos genero automáticamente una estructura tipo tabla:



16. Por ultimo para poder darle un estilo diferente a cada input del formulario creado desde Django en **forms.py** desde ahí mismo podemos pasarle los atributos **id**, **class** y todas propiedades que necesitemos. Se recomienda en este caso pasar los selectores y luego desde css darle el estilo deseado.

Entonces para eso usamos los **widget** de Django, para mayor referencia ver:

<https://docs.djangoproject.com/en/3.2/ref/forms/widgets/>

Hacemos la siguiente modificación en **forms.py** y vemos que desde django podemos darle estilo a nuestro front:

from django import forms

class FormularioRemera(forms.Form):

    marca = forms.CharField(label="Marca", max\_length=128, widget=forms.TextInput(attrs={'class' : 'inputTexto', 'id': 'marca', 'style': 'background-color: red;'}))

    TALLES = (

        (1, "XS"),

        (2, "S"),

        (3, "M"),

        (4, "L"),

        (5, "XL"),

        (6, "XXL")

    )

    talle = forms.ChoiceField(label="Talle", choices=TALLES, widget=forms.Select(attrs={'class' : 'inputTalle', 'id': 'talle'}))

    color = forms.CharField(label="Color", max\_length=128, widget=forms.TextInput(attrs={'class' : 'inputColor', 'size': '40', 'id': 'color'}))

    lisa = forms.BooleanField(label="Lisa", required=False, widget=forms.CheckboxInput( attrs={'class' : 'inputLisa'}))

    GENEROS = (

        (1, "Hombre"),

        (2, "Mujer"),

        (3, "Unisex")

    )

    genero = forms.ChoiceField(label="Genero", choices=GENEROS)

Y obtenemos la salida:

A su vez, aprovechando los id podemos darle mayor estilo desde css

**Estilos.css**

.inputTexto{

    font-size: 30px;

}

#talle{

    color: aqua;

}

#color{

    background-color: brown;

}

#genero\_id{

    margin-left: 10px;

}

Recordemos linkear la hoja de estilos a **base.html**

{% load static %}

<html>

<head>

    <title>

        {% block title %} {% endblock %}

    </title>

    <link rel="stylesheet" href='{% static "shop/estilos.css" %}'>

</head>

<a href="{% url 'index' %}">

    <img src="{% static 'shop/logo.png'  %}" >

</a>

<body>

   {% block content %}{% endblock %}  <!--generamos un bloque lo que significa que en las

                                        plantillas que extienden o heredan de base.html

                                         se aplicara codigo html donde se marque el bloque

                                         -->

</body>

</html>

Veremos la siguiente salida:



Modificaciones necesarias para el talle, Lisa y Genero.

**Talle**

   <td>

                {% if talle == 1 %}

                    XS

                {% endif %}

                {% if talle == 2 %}

                    S

                {% endif %}

                {% if talle == 3 %}

                    L

                {% endif %}

                {% if talle == 4 %}

                    M

                {% endif %}

                {% if talle == 5 %}

                    XL

                {% endif %}

                {% if talle == 6 %}

                    XXL

                {% endif %}

            </td>

**Lisa**

 <td>

                {% if lisa  %}

                    SI

                {% else %}

                    NO

                {% endif %}

            </td>

**Genero Se propone para la clase que viene colocar el condicional necesario para solucionar esta variable.**

**CheckPoint Codigo:**

****