

Django - parte 4

Programación Web 2

Objetos Form “planos” en Django

- Otra forma de form es el básico

- `vim personas/forms.py`
- Observe que
 - ya no heredamos de **ModelForm**
 - ¿El formulario tiene los mismos campos y tipos que el modelo?

```
class RawPersonaForm(forms.Form):  
    nombres = forms.CharField()  
    apellidos = forms.CharField()  
    edad = forms.IntegerField()  
    donador = forms.BooleanField()
```

- Crearemos una nueva vista para este form y la registraremos

- `vim personas/views.py`
- `vim listaContactos/urls.py`
- El template no se modifica

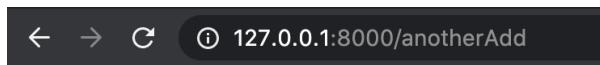
```
from personas.views import personasTestView, personaCreateView, personasAnotherCreateView  
  
urlpatterns = [  
    path('', myHomeView, name='PaginaInicio'),  
    path('people', personasTestView, name='personas'),  
    path('add', personaCreateView, name='AgregarPersonas'),  
    path('anotherAdd', personasAnotherCreateView, name='OtroAgregarPersonas'),  
    path('another', anotherView, name='otra'),  
    path('admin/', admin.site.urls),  
]
```

```
from .forms import PersonaForm, RawPersonaForm  
  
# Create your views here.  
def personasAnotherCreateView(request):  
    form = RawPersonaForm()  
    context = {  
        'form': form,  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

Revise si algún tipo de dato permitido en el modelo no existe en los tipos de datos de forms

Parece que el formulario funciona, pero NO

La base de datos ¡no se modificó!



- Home
- Primera
- Segunda
- Tercera

```
>>> from personas.models import Persona
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (1)>, <Persona: Persona object (2)>, <Persona: Persona object (3)>, <Persona: Persona object (4)>]>
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (1)>, <Persona: Persona object (2)>, <Persona: Persona object (3)>, <Persona: Persona object (4)>]>
```

Usando un template Base

Nombres:

Apellidos:

Edad:

Donador: ☒

Inspeccione el código
HTML generado, observe
los tags generados

```
{% extends 'base.html' %}
{% block content %}
<form method='POST'> {% csrf_token %}
  {{ form.as_p }}
  <input type='submit' value='Grabar' />
</form>
{% endblock %}
```

Modifique el template y

Explore otras opciones distintas a form.as_p

<https://docs.djangoproject.com/en/3.2/ref/forms/api/#as-p>

Enviando los datos al formulario

El primer paso para poder grabar los datos en la base de datos es enviar los datos al objeto formulario, esto sólo se **debe** hacer para una llamada POST y no para GET

```
def personasAnotherCreateView(request):  
    form = RawPersonaForm(request.POST)  
    context = {  
        'form': form,  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

← → ↺ ⓘ 127.0.0.1:8000/anotherAdd

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

- Este campo es obligatorio.

Nombres:

- Este campo es obligatorio.

Apellidos:

- Este campo es obligatorio.

Edad:

- Este campo es obligatorio.

Donador: ☐

Diferenciando llamada POST y GET: Seguridad




- Home
- Primera
- Segunda
- Tercera

```
def personasAnotherCreateView(request):  
    form = RawPersonaForm() #request.GET  
    if request.method == "POST":  
        form = RawPersonaForm(request.POST)  
    context = {  
        'form': form,  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

Usando un template Base

Nombres:

Apellidos:

Edad: 2  Completa este campo

Donador: ☒

Ponga a prueba la seguridad de Django haciendo algo de hacking sobre el código HTML: haga que los campos no sean "requeridos"

Recibiendo data limpia en una vista

El objeto form, cuenta con elementos que permitirán evaluar si los datos son válidos, estos chequeos se realizan a nivel del formulario y del servidor

```
def personasAnotherCreateView(request):  
    form = RawPersonaForm() #request.GET  
    if request.method == "POST":  
        form = RawPersonaForm(request.POST)  
        if form.is_valid():  
            print(form.cleaned_data)  
        else:  
            print(form.errors)  
    context = {  
        'form': form,  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

```
{'nombres': '123', 'apellidos': 'Paz', 'edad': 47, 'donador': True}  
[28/Jun/2021 11:19:43] "POST /add/ HTTP/1.1" 200 974
```

Haga un poco de hacking sobre el código HTML, cambie el tipo de dato de los campos en el formulario

Finalmente grabamos los datos

Los dos `**` delante de un diccionario significan: "tratar los pares clave-valor en el diccionario como argumentos nombrados adicionales para esta llamada de función".

```
def personasAnotherCreateView(request):  
    form = RawPersonaForm() #request.GET  
    if request.method == "POST":  
        form = RawPersonaForm(request.POST)  
        if form.is_valid():  
            print(form.cleaned_data)  
            Persona.objects.create(**form.cleaned_data)  
        else:  
            print(form.errors)  
    context = {  
        'form': form,  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

<https://stackoverflow.com/questions/21809112/what-does-tuple-and-dict-means-in-python>

← → ↻ ⓘ 127.0.0.1:8000/anotherAdd

- Home
- Primera
- Segunda
- Tercera

Tanto Form como RawForm hacen un trabajo equivalente, serían casi intercambiables, la principal diferencia está cómo se usan en VIEW

Usando un template Base

Nombres:

Apellidos:

Edad:

Donador: ☒

```
(django) apaz src $ python manage.py shell
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from personas.models import Persona
>>> Persona.objects.all()
<QuerySet [<Persona: Persona object (1)>, <Persona: Persona object (2)>, <Persona: Persona object (3)>, <Persona: Persona object (4)>, <Persona: Persona object (5)>, <Persona: Persona object (6)>]>
>>> obj = Persona.objects.get(id = 6)
>>> obj.nombres
'Otro 2'
```

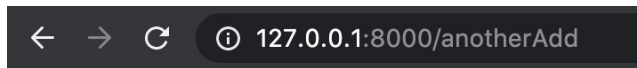

Campos en el formulario

En un formulario, cada campo tiene una lógica de validación personalizada, junto con algunos otros hooks.

Las opciones llamadas “core” están presentes en todos los tipos de campos.

```
class RawPersonaForm(forms.Form):  
    nombres = forms.CharField(label = 'Your Name')  
    apellidos = forms.CharField()  
    edad = forms.IntegerField(initial = 20)  
    donador = forms.BooleanField()
```

¿Cuál es el valor para las opciones core **by default**?



- Home
- Primera
- Segunda
- Tercera

Usando un template Base

Your Name:

Apellidos:

Edad:

Donador: ☐

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

Widgets

Un widget es la representación de Django de un elemento de input HTML. El widget maneja la representación del HTML y la extracción de datos de un diccionario GET / POST que corresponde al widget.

```
class RawPersonaForm(forms.Form):
    nombres = forms.CharField(
        widget = forms.Textarea(
            attrs={
                'placeholder': 'Sólo tu nombre, por favor',
                'id': 'nombreID',
                'class': 'special',
                'cols': '10',
            }
        )
    )
    apellidos = forms.CharField()
    edad = forms.IntegerField()
    donador = forms.BooleanField()
```

← → ↺ ⓘ 127.0.0.1:8000/anotherAdd

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

Sólo tu nombre, por favor

Nombres:

Apellidos:

Edad:

Donador: ☐

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

Validación extra de campos: `clean_<fieldname>()`

- La validación de un form ocurre cuando los datos están limpios.
- En general, cualquier método de limpieza puede lanzar **ValidationError**
- Cada campo en una clase de Form es responsable no solo de validar los datos, sino también de "limpiarlos", normalizándolos a un formato consistente (`cleaned_data`).

```
class PersonaForm(forms.ModelForm):
    class Meta:
        model = Persona
        fields = [
            'nombres',
            'apellidos',
            'edad',
            'donador',
        ]
    def clean_nombres(self, *args, **kwargs):
        print('paso')
        name = self.cleaned_data.get('nombres')
        if name.istitle():
            return name
        else:
            raise forms.ValidationError('La primera letra en mayúscula')
```

← → ↻ ⓘ 127.0.0.1:8000/add

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

- La primera letra en mayúscula

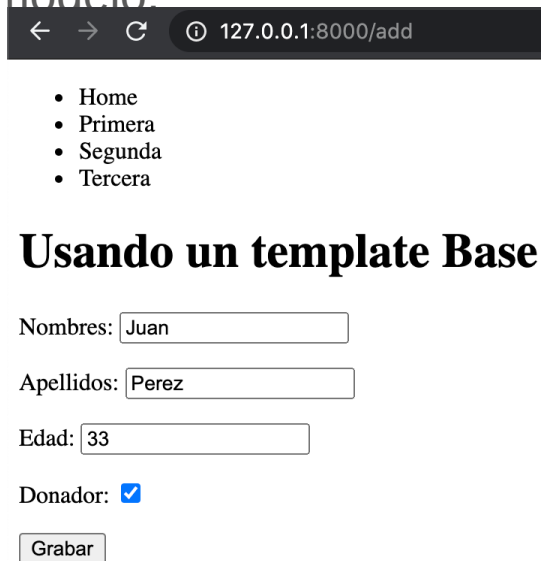
Nombres:

Datos de iniciales de una instancia

En algún caso se podría desear que el formulario contenga datos que ya están en nuestro modelo (por ejemplo en un blog). La opción `instance`, permite llenar el formulario con datos iniciales desde un objeto del modelo.

```
def personaCreateView(request):  
    obj = Persona.objects.get(id = 2)  
    form = PersonaForm(request.POST or None, instance = obj)  
    if form.is_valid():  
        form.save()  
        form = PersonaForm()  
    context = {  
        'form': form  
    }  
    return render(request, 'personas/personasCreate.html', context)
```

Esto permite editar un objeto, ¡Pruébalo!



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/add`. Below the address bar is a list of navigation links: Home, Primera, Segunda, and Tercera. The main content area displays a form with the following fields: 'Nombres' with the value 'Juan', 'Apellidos' with the value 'Perez', 'Edad' with the value '33', and 'Donador' with a checked checkbox. At the bottom of the form is a button labeled 'Grabar'.

Valores iniciales para un campo del formulario

Los valores iniciales se pueden proporcionar con el parámetro **initial=** estos anularán tanto los valores iniciales del campo de formulario como los valores de una instancia de modelo adjunta

```
def personaCreateView(request):
    initialValues = {
        'nombres' : 'Sin Nombre'
    }
    form = PersonaForm(request.POST or None, initial = initialValues)
    if form.is_valid():
        form.save()
        form = PersonaForm()
        context = {
            'form': form
        }
    return render(request, 'personas/personasCreate.html', context)
```

Usando un template Base

Nombres:

Apellidos:

Edad:

Donador: ☐

<https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/#providing-initial-values>

Ruteo dinámico de URL

- Se desea navegar sobre los elementos almacenados en nuestro modelo.
 - La ruta asociada a una vista puede contener valores dinámicos <int: myId>

```
from personas.views import (
    personasTestView,
    personaCreateView,
    personasShowObject,
)

urlpatterns = [
    path('', myHomeView, name='PaginaInicio'),
    path('people', personasTestView, name='personas'),
    path('add', personaCreateView, name='AgregarPersonas'),
    path('another', anotherView, name='otra'),
    path('admin/', admin.site.urls),
    path('personas/<int:myID>/', personasShowObject, name = 'browsing')
]
```

```
def personasShowObject(request, myID):
    obj = Persona.objects.get(id = myID)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/descripcion.html', context)
```

← → ↺ ⓘ 127.0.0.1:8000/personas/3/

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

Jorge Gonzales

18 años

```
{% extends 'base.html' %}

{% block content %}
<h1>{{ objeto.nombres }} {{ objeto.apellidos }}</h1>
<h2>{{ objeto.edad }} años</h2>
{% endblock %}
```

<https://docs.djangoproject.com/en/3.2/topics/http/urls/>

Error 404: get_object_or_404

Al navegar en los urls dinámicos podemos llegar a un objeto que no existe.

```
from django.shortcuts import render, get_object_or_404
from .models import Persona
from .forms import PersonaForm

# Create your views here.
def personasShowObject(request, myID):
    obj = get_object_or_404(Persona, id = myID)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/descripcion.html', context)
```

← → ↺ ⓘ 127.0.0.1:8000/personas/37/

Page not found (404)

Request Method: GET

Request URL: http://127.0.0.1:8000/personas/37/

Raised by: personas.views.personasShowObject

<https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#get-object-or-404>

Borrando Objetos

- Se tiene que hacer en una llamada POST
- El método delete()

```
def personasDeleteView(request, myID):  
    obj = get_object_or_404(Persona, id = myID)  
    if request.method == 'POST':  
        print("lo borro")  
        obj.delete()  
        context = {  
            'objeto': obj,  
        }  
    return render(request, 'personas/personasBorrar.html', context)
```

```
path('personas/<int:myID>/delete/', personasDeleteView, name = 'deleting'),
```

```
{% extends 'base.html' %}  
{% block content %}  
  
<form action='.' method='POST'> {% csrf_token %}  
  <h1>Quiere borrar a esta persona: </h1>  
  <h2>{{ objeto.nombres }}</h2>  
  <input type='submit' value='Si' />  
  <a href='..'>Cancelar</a>  
</form>  
  
{% endblock %}
```

← → ↻ ⓘ 127.0.0.1:8000/personas/4/delete/

Page not found (404)

Request Method: GET

Request URL: http://127.0.0.1:8000/personas/4/delete/

Raised by: personas.views.personasDeleteView

<https://docs.djangoproject.com/en/3.2/ref/models/instances/#deleting-objects>

Redireccionando

```
from django.shortcuts import render, get_object_or_404, redirect
from .models import Persona
from .forms import PersonaForm

# Create your views here.
def personasDeleteView(request, myID):
    obj = get_object_or_404(Persona, id = myID)
    if request.method == 'POST':
        print("lo borro")
        obj.delete()
        return redirect('../')
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/personasBorrar.html', context)
```

<https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#redirect>

Listando todos los objetos

```
def personasListView(request):  
    queryset = Persona.objects.all()  
    context = {  
        'objectList': queryset,  
    }  
    return render(request, 'personas/personasLista.html', context)
```

```
{% extends 'base.html' %}  
  
{% block content %}  
  
{% for instance in objectList %}  
<p> {{ instance.id }} - {{ instance.nombres }}</p>  
{% endfor %}  
  
{% endblock %}
```

```
path('personas/', personasListView, name = 'listing'),
```

← → ↻ ⓘ 127.0.0.1:8000/personas/

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

1 - Alfredo

2 - Juan

3 - Jorge

Enlaces dinámicos de URL

Defina un método `get_absolute_url()` para decirle a Django cómo calcular la URL canónica de un objeto.

```
class Persona(models.Model):
    nombres = models.CharField(max_length = 100)
    apellidos = models.CharField(max_length = 100)
    edad = models.IntegerField()
    donador = models.BooleanField()

    def get_absolute_url(self):
        return "/personas/" + str(self.id) + "/"
```

```
{% extends 'base.html' %}

{% block content %}

{% for instance in objectList %}
<p>
    {{ instance.id }}
    -
    <a href='{{ instance.get_absolute_url }}'>
        {{ instance.nombres }}
    </a>
</p>
{% endfor %}

{% endblock %}
```

https://docs.djangoproject.com/en/3.2/ref/models/instances/#django.db.models.Model.get_absolute_url

Invirtiendo los URLs: absolute_url

Una necesidad común cuando se trabaja en un proyecto de Django es la posibilidad de obtener URL en sus formas finales, ya sea para incrustar en el contenido generado (URL de vistas y activos, URL mostradas al usuario, etc.) o para manejar el flujo de navegación en el lado del servidor (redirecciones, etc.)

```
from django.db import models
from django.urls import reverse

# Create your models here.
class Persona(models.Model):
    nombres = models.CharField(max_length = 100)
    apellidos = models.CharField(max_length = 100)
    edad = models.IntegerField()
    donador = models.BooleanField()

    def get_absolute_url(self):
        return reverse('browsing', kwargs={'myID': self.id})
```

<https://docs.djangoproject.com/en/3.2/ref/urlresolvers/#django.urls.reverse>

Espacio de nombres

Se creará un archivo para que la misma app maneje sus URLs

```
from django.urls import path
from personas.views import (
    personasTestView,
    personaCreateView,
    personasShowObject,
    personasDeleteView,
    personasListView,
)

app_name = 'personas'
urlpatterns = [
    path('add/', personaCreateView, name='adding'),
    path('<int:myID>/', personasShowObject, name = 'browsing'),
    path('<int:myID>/delete/', personasDeleteView, name = 'deleting'),
    path('', personasListView, name = 'listing'),
]
```

```
from django.contrib import admin
from django.urls import path, include
from inicio.views import myHomeView, anotherView

urlpatterns = [
    path('personas/', include('personas.urls')),
    path('', myHomeView, name='PaginaInicio'),
    path('another', anotherView, name='otra'),
    path('admin/', admin.site.urls),
]
```

```
personas
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   ├── admin.cpython-37.pyc
│   ├── forms.cpython-37.pyc
│   ├── models.cpython-37.pyc
│   ├── urls.cpython-37.pyc
│   └── views.cpython-37.pyc
├── admin.py
├── ...
├── tests.py
├── urls.py
└── views.py
```

```
from django.db import models
from django.urls import reverse

# Create your models here.
class Persona(models.Model):
    nombres = models.CharField(max_length = 100)
    apellidos = models.CharField(max_length = 100)
    edad = models.IntegerField()
    donador = models.BooleanField()

    def get_absolute_url(self):
        return reverse('personas:browsing', kwargs={'myID': self.id})
```