

Django - parte 3

Programación Web 2

El “contexto”

- La idea es que las vistas proporcionen “datos” a los templates, para que estos los puedan mostrar.
- Los templates deben crear código html y los datos enviados en el “contexto” (por una vista) para producir un resultado visible.

```
# Create your views here.  
def myHomeView(request, *args, **kwargs):  
    return render(request, 'home.html', {})
```



Las variables del “contexto” en un template

- Para enviar variables desde la vista, hay que crear un diccionario y enviarlo a la función render.
- En el template, los elementos del diccionario se tratan como variables.

```
def myHomeView(request, *args, **kwargs):  
    myContext = {  
        'myText': 'Esto es sobre nosotros',  
        'myNumber': 123,  
    }  
    return render(request, 'home.html', myContext)
```

```
{% extends 'base.html' %}  
{% block content %}  
<h2>Hola Mundo desde Django</h2>  
<h3>Con templates</h3>  
<p>  
    {{myText}},  
    {{myNumber}}  
</p>  
{% endblock %}
```



Este es un texto Base

Hola Mundo desde Django

Con templates

Esto es sobre nosotros, 123

El tag: FOR

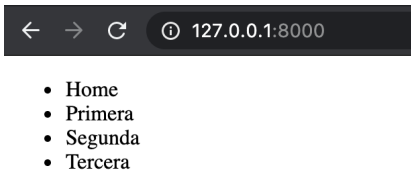
<https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#for>

- Los datos enviados al template, no tienen que ser simples, pueden ser listas!
- Para trabajar con datos compuestos, se cuenta con el tag

FOR

```
{% extends 'base.html' %}
{% block content %}
<h2>Hola Mundo desde Django</h2>
<h3>Con templates</h3>
<p>
    {{myText}},
    {{myNumber}}
</p>
<ul>
    {% for myItem in myList %}
        <li>{{ myItem }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

```
def myHomeView(request, *args, **kwargs):
    myContext = {
        'myText': 'Esto es sobre nosotros',
        'myNumber': 123,
        'myList': [33, 44, 55],
    }
    return render(request, 'home.html', myContext)
```



Este es un texto Base

Hola Mundo desde Django

Con templates

Esto es sobre nosotros, 123

- 33
- 44
- 55

El contador de ciclos

- El tag for cuenta con algunas variables predefinidas como `forloop.counter`

```
{% extends 'base.html' %}
{% block content %}
<h2>Hola Mundo desde Django</h2>
<h3>Con templates</h3>
<p>
    {{myText}},
    {{myNumber}}
</p>
<ul>
    {% for myItem in myList %}
    <li>{{forloop.counter}} - {{ myItem }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

Revisar el objeto `forloop` en la documentación oficial y hacer experimentos



- Home
- Primera
- Segunda
- Tercera

Este es un texto Base

Hola Mundo desde Django

Con templates

Esto es sobre nosotros, 123

- 1 - 33
- 2 - 44
- 3 - 55

El tag: IF

El DTL también posee condicionales

```
{% extends 'base.html' %}
{% block content %}
<h2>Hola Mundo desde Django</h2>
<h3>Con templates</h3>
<p>
  {{myText}},
  {{myNumber}}
</p>
<ul>
  {% for myItem in myList %}
    <li>{{forloop.counter}} - {{ myItem }}
    {% if myItem == 33 %}
      - Este número es mágico
    {% elif myItem > 50 %}
      - Este número es mayor que 50
    {% endif %}
    </li>
  {% endfor %}
</ul>
{% endblock %}
```

<https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#if>

Revisar la documentación oficial sobre if, hacer experimentos con otros operadores de comparación



- Home
- Primera
- Segunda
- Tercera

Este es un texto Base

Hola Mundo desde Django

Con templates

Esto es sobre nosotros, 123

- 1 - 33 - Este número es mágico
- 2 - 44
- 3 - 55 - Este número es mayor que 50

Filtros

<https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#built-in-filter-reference>

- Los filtros pueden ser aplicados a bloques o variables
- Pueden ser “compuestos” usando el operador pipe |

```
{% extends 'base.html' %}
{% block content %}
<h2>Hola Mundo desde Django</h2>
<h3>Con templates</h3>
<p>
  {{myText|upper}},
  {{myNumber}}
</p>
<ul>
  {% for myItem in myList %}
    <li>{{forloop.counter}} - {{ myItem }}
    {% if myItem == 33 %}
      - Este número es mágico
    {% elif myItem > 50 %}
      - Este número es mayor que 50
    {% endif %}
    </li>
  {% endfor %}
</ul>
{% endblock %}
```

← → ↺ 127.0.0.1:8000

- Home
- Primera
- Segunda
- Tercera

Este es un texto Base

Hola Mundo desde Django

Con templates

ESTO ES SOBRE NOSOTROS, 123

- 1 - 33 - Este número es mágico
- 2 - 44
- 3 - 55 - Este número es mayor que 50

Probar la aplicación de varios otros filtros a variables y bloques, revisando la documentación oficial

Más ejemplos de Filtros

```
{% extends 'base.html' %}
{% block content %}
<h2>Hola Mundo desde Django</h2>
<h3>Con templates</h3>
<p>
  {{myText|upper|truncatechars:"16"}} ,
  {{myNumber}}
</p>
<ul>
  {% for myItem in myList %}
    <li>{{forloop.counter}} - {{ myItem|add:2 }}
    {% if myItem == 33 %}
      - Este número es mágico
    {% elif myItem|divisibleby:2 %}
      - Este número es par
    {% elif myItem > 50 %}
      - Este número es mayor que 50
    {% endif %}
    </li>
  {% endfor %}
</ul>
{% endblock %}
```

← → ↺ ⓘ 127.0.0.1:8000

- Home
- Primera
- Segunda
- Tercera

Este es un texto Base

Hola Mundo desde Django

Con templates

ESTO ES SOBRE..., 123

- 1 - 35 - Este número es mágico
- 2 - 46 - Este número es par
- 3 - 57 - Este número es mayor que 50

Examinando los objetos del Modelo

Los objetos del modelo contienen los campos que definimos y algunos otros más heredados del framework

```
from django.db import models

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

```
(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)>, <Person
a: Persona object (3)>]>
[>>> obj = Persona.objects.get(id = 2)
[>>> dir(obj)
['_DoesNotExist', 'MultipleObjectsReturned', '__class__', '__delattr__', '__dict__
__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__
getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__
repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook
__', '__weakref__', '_check_column_name_clashes', '_check_field_name_clashes', '_
_check_fields', '_check_id_field', '_check_index_together', '_check_indexes', '_
_check_local_fields', '_check_long_column_names', '_check_m2m_through_same_relati
onship', '_check_managers', '_check_model', '_check_model_name_db_lookup_clashes
', '_check_ordering', '_check_property_name_related_field_accessor_clashes', '_c
heck_single_primary_key', '_check_swappable', '_check_unique_together', '_do_ins
ert', '_do_update', '_get_FIELD_display', '_get_next_or_previous_by_FIELD', '_ge
t_next_or_previous_in_order', '_get_pk_val', '_get_unique_checks', '_meta', '_pe
rform_date_checks', '_perform_unique_checks', '_save_parents', '_save_table', '_
set_pk_val', '_state', 'apellidos', 'check', 'clean', 'clean_fields', 'date_erro
r_message', 'delete', 'donador', 'edad', 'from_db', 'full_clean', 'get_deferred_
fields', 'id', 'nombres', 'objects', 'pk', 'prepare_database_save', 'refresh_fro
m_db', 'save', 'save_base', 'serializable_value', 'unique_error_message', 'valid
ate_unique']
```

Renderizando una Persona de la app personas

Se requieren dos cosas: una vista y un template

- `vim` `personas/views.py`
- `mkdir` `templates/personas`
- `vim` `templates/personas/test.html`
- `vim` `listaPersonas/urls.py`

```
from django.contrib import admin
from django.urls import path
from inicio.views import myHomeView
from inicio.views import anotherView
from personas.views import personaTestView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', myHomeView, name='home'),
    path('another/', anotherView, name='otro'),
    path('persona/', personaTestView, name='otro'),
]
```

```
from django.shortcuts import render
from .models import Persona

# Create your views here.
def personaTestView(request):
    obj = Persona.objects.get(id = 1)
    context = {
        'nombre': obj.nombres,
        'edad': obj.edad,
    }
    return render(request, 'personas/test.html', context)
```

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

{% block content%}
<h1>{{nombre}}</h1>
<h2>
    {% if edad > 40 %}
    debe cuidarse del colesterol
    {% else %}
    puede comer lo que quiere
    {% endif %}
</h2>
{% endblock %}
```

← → ↺ ⓘ 127.0.0.1:8000/persona/

- Home
- Primera
- Segunda
- Tercera

Este es un texto Base

Alfredo

puede comer lo que quiera

Enviando el objeto entero en el contexto

- En lugar de crear un diccionario en la vista con los detalles del objeto, también se puede enviar el objeto entero:

```
from django.shortcuts import render
from .models import Persona

# Create your views here.
def personaTestView(request):
    obj = Persona.objects.get(id = 1)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/test.html', context)
```

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

{% block content%}
<h1>{{ objeto.nombres }}</h1>
<h2>
    {% if objeto.edad > 40 %}
    debe cuidarse del colesterol
    {% else %}
    puede comer lo que quiera
    {% endif %}
</h2>
{% endblock %}
```

Independizando la app personas

- Una app en Django debe ser suficientemente independiente como para que pueda ser usada en otros proyectos o incluso comercializada como producto separado.
- El estar usando un directorio externo para las plantillas de la app personas, no permite que el app sea independiente del proyecto.
- Python tiene un sistema de búsqueda de plantillas, que siempre buscará el directorio “templates” dentro del directorio del app
 - src/templates/personas/descripcion.html (de TEMPLATES = 'DIR': en settings.py)
 - python3/... contrib/admin/templates/personas/descripcion.html
 - python3/... contrib/auth/templates/personas/descripcion.html
 - src/personas/templates/personas/descripcion.html

Creando la plantilla dentro del app Persona

- Un directorio para las plantillas dentro del app
 - `mkdir personas/templates`
- Se mueve el directorio de templates para el app
 - `mv templates/personas personas/templates`
- Se renombra la plantilla para la app
 - `mv personas/templates/personas/test.html personas/templates/personas/descripcion.html`
- `vim personas/views.py`

```
from django.shortcuts import render
from .models import Persona

# Create your views here.
def personaTestView(request):
    obj = Persona.objects.get(id = 1)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/descripcion.html', context)
```

```
personas
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   ├── admin.cpython-37.pyc
│   ├── models.cpython-37.pyc
│   └── views.cpython-37.pyc
├── admin.py
├── apps.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_auto_20200612_1449.py
│   ├── 0003_persona_donador.py
│   ├── 0004_auto_20200612_1457.py
│   ├── __init__.py
│   └── __pycache__
│       ├── 0001_initial.cpython-37.pyc
│       ├── 0002_auto_20200612_1449.cpython-37.pyc
│       ├── 0003_persona_donador.cpython-37.pyc
│       └── 0004_auto_20200612_1457.cpython-37.pyc
├── models.py
├── templates
│   └── personas
│       └── descripcion.html
├── tests.py
└── views.py
```

Formularios en Django

<https://docs.djangoproject.com/es/3.2/topics/forms/>

- El ingreso de nuevos datos al modelo necesita su propio formulario.
 - La tarea de un formulario no es sencilla. Django maneja:
 - La preparación y reestructuración de los datos para dejarlos listos para el rendering.
 - La creación de formularios HTML para los datos
 - Cómo recibir (seguridad) y procesar formularios y datos enviados por el cliente.
- La clases de tipo Form
 - El Modelo se encarga de los datos, El Form, se encargan de los formularios: como funciona y como se muestra
 - La clase ModelForm, mapea los campos del modelos a elementos `<input>` de un formulario HTML a través de un Form

Formulario sencillo para el app Personas

- **vim** personas/forms.py
 - Nuestra clase debe heredar de ModelForm
 - En la clase Meta se indica el modelo asociado al Formulario
 - En el arreglo fields
 - Se establece que campos se incluirán y en que orden
- **vim** personas/views.py
 - Se instancia un objeto de nuestra clase (Form)
 - indicando si se usará POST o GET
 - Se verifica que el formulario sea válido
 - is_valid()
 - Se graban los datos del formulario
 - save()
 - se limpia el formulario luego de grabar

```
from django import forms

from .models import Persona

class PersonaForm(forms.ModelForm):
    class Meta:
        model = Persona
        fields = [
            'nombres',
            'apellidos',
            'edad',
            'donador',
        ]
```

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

# Create your views here.
def personaTestView(request):
    obj = Persona.objects.get(id = 1)
    context = {
        'objeto': obj,
    }
    return render(request, 'personas/descripcion.html', context)

def personaCreateView(request):
    form = PersonaForm(request.POST or None)
    if form.is_valid():
        form.save()
        form = PersonaForm()

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

Intente llenar el formulario para hacerlo inválido

Formulario sencillo para el app Personas

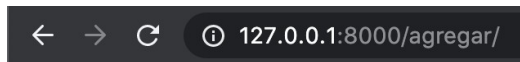
- **vim** personas/templates/personas/personasCreate.html
 - En la plantilla se dibujan los campos usando as_p
 - csrf_token es necesario para la conexión POST
- **vim** listaContactos/urls.py
 - Hay que asociar la vista con algún URL

```
from inicio.views import anotherView
from personas.views import personaTestView, personaCreateView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', myHomeView, name='home'),
    path('another/', anotherView, name='otro'),
    path('persona/', personaTestView, name='testViewPersona'),
    path('agrega/', personaCreateView, name='createPersona'),
]
```

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


El formulario creado funcionando



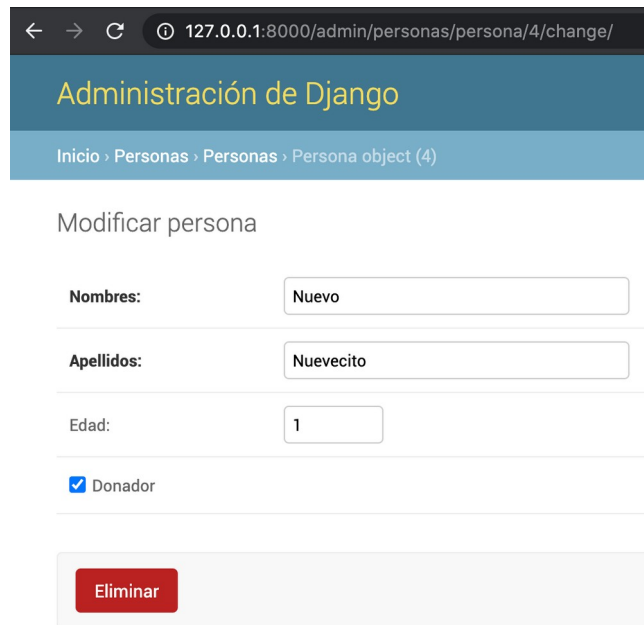
Este es un texto Base

Nombres:

Apellidos:

Edad:

Donador: ☒



← → ↻ ⓘ 127.0.0.1:8000/admin/personas/persona/4/change/

Administración de Django

Inicio › Personas › Personas › Persona object (4)

Modificar persona

Nombres:

Apellidos:

Edad:

☒ Donador

Pruebe qué ocurre si no incluimos el campo “donador” en el formulario

¿De qué se trata este error?

¿Cómo se puede solucionar?

GET y POST

Recordemos las características de estos métodos:

- GET envía los datos usando el URL
- GET NO debe ser usado:
 - para modificar el estado del sistema (modelos).
 - para enviar un password, este se verá en el URL, en los logs del servidor y se almacenará en el historial del navegador.
 - para formularios de administración,
- POST permite enviar mayor cantidad de información que GET
- POST con el apoyo de otras herramientas como la protección de Cross Site Request Forgery ([csrf](#)) de Django, ofrece mayor seguridad

Note que una de nuestras vistas se invoca con GET y luego con POST

Templates con forms planos: GET

- Creamos un nuevo template “plano” que use GET

- `vim personas/templates/personas/search.html`

```
{% extends 'base.html' %}
{% block content %}
<form action='http://docs.djangoproject.com/en/2.1/search/' method='GET'>
  <input type='text' name='q' placeholder='buscar' />
  <input type='submit' value='Buscar' />
</form>
{% endblock %}
```

- Creamos una nueva vista y la asociamos a un URL

```
def searchForHelp(request):
    return render(request, 'personas/search.html', {})
```

```
from personas.views import personasTestView, personaCreateView, searchForHelp

urlpatterns = [
    path('', myHomeView, name='PaginaInicio'),
    path('people', personasTestView, name='personas'),
    path('add', personaCreateView, name='AgregarPersonas'),
    path('search', searchForHelp, name='buscar'),
    path('another', anotherView, name='otra'),
    path('admin/', admin.site.urls),
]
```

← → ↺ ⓘ 127.0.0.1:8000/search

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

buscar

Datos de GET y POST

- `vim personas/templates/personas/personasCreate.html`

```
{% extends 'base.html' %}
{% block content %}
<form action='add' method='GET'>
  <input type='text' name='q' placeholder='buscar' />
  <input type='submit' value='Buscar' />
</form>
{% endblock %}
```

cambiar action='.'

- `vim personas/views.py`

```
def personaCreateView(request):
    print('GET: ', request.GET)
    print('POST: ', request.POST)
    context = {}
    return render(request, 'personas/personasCreate.html', context)
```

```
[24/Jun/2020 11:08:49] "GET /add HTTP/1.1" 200 425
GET:  <QueryDict: {'q': ['Jorge']}>
POST:  <QueryDict: {}>
```

← → ↻ ⓘ 127.0.0.1:8000/add?q=Jorge

- Home
- Primera
- Segunda
- Tercera

Usando un template Base

Templates con forms planos: POST

- `vim` `personas/templates/personas/personasCreate.html`

```
{% extends 'base.html' %}
{% block content %}
<form action='add' method='POST'>
  <input type='text' name='q' placeholder='buscar' />
  <input type='submit' value='Buscar' />
</form>
{% endblock %}
```

← → ↻ ⓘ 127.0.0.1:8000/agregar/

Prohibido (403)

Verificación CSRF fallida. Solicitud abortada

- Cuando se usa post Django obliga usar el `csrf_token`

```
{% extends 'base.html' %}
{% block content %}
<form action='add' method='POST'> {% csrf_token %}
  <input type='text' name='q' placeholder='buscar' />
  <input type='submit' value='Buscar' />
</form>
{% endblock %}
```

```
[24/Jun/2020 11:28:38] "GET /add?q=Jorge HTTP/1.1" 200 547
GET: <QueryDict: {}>
POST: <QueryDict: {'csrfmiddlewaretoken': ['hxE0W3zLZMyABJQTQR3qDujhRjns5o02mbDYfmUucv8IV0F5omfwrypQU3moR10'], 'q': ['Jorge']]>
```

- <https://docs.djangoproject.com/en/3.2/ref/csrf/>

Acciones distintas para GET y POST

- Del ejemplo anterior vemos que la vista se llama primero con GET y luego con POST ¿En qué momento se hace llamada GET y en qué momento se hace la llamada POST?
 - `vim personas/views.py`

```
def personaCreateView(request):  
    print(request)  
    if request.method == 'POST':  
        nombre = request.POST.get('q')  
        print(nombre)  
        context = {}  
        return render(request, 'personas/personasCreate.html', context)
```

```
[24/Jun/2020 11:36:17] "GET /admin/login/?next=/admin/ HTTP/1.1" 200 1872  
<WSGIRequest: GET '/add'>  
[24/Jun/2020 11:36:17] "GET /add HTTP/1.1" 200 547  
<WSGIRequest: POST '/add'>  
Jorge  
[24/Jun/2020 11:36:26] "POST /add HTTP/1.1" 200 547
```