Django - parte 5

Programación Web 2

Vistas basadas en clases

- Una vista, de manera similar a un CGI, toma una solicitud y devuelve una respuesta.
- Django proporciona algunas clases que se pueden usar como base de nuestras propias vistas.
- Estas clases, le permitirán reutilizar código aprovechando la herencia y los mixins.
- Hay algunas vistas genéricas para tareas simples, estas permiten:
  - Listar elementos - ListView
  - Ver el detalle de un objeto - DetailView
  - Agregar objetos - CreateView
  - Modificar objetos - UpdateView
  - Borrar objetos - DeleteView

Usando ListView

Como veremos el uso de estas vistas supone la existencia de algunos archivos con nombres específicos.

- vim personas/views.py
- vim personas/urls.py
- vim listaContactos/urls.py

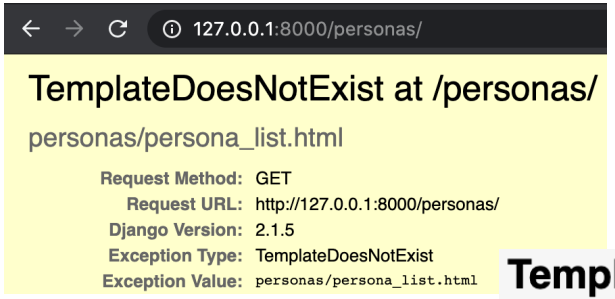
```python
from django.shortcuts import render
from django.views.generic.list import (
    ListView,
    )
from .models import Persona

# Create your views here.
class PersonaListView(ListView):
  model = Persona
```

```python
from django.urls import path
from .views import (
    PersonaListView,
    )

urlpatterns = [
    path('', PersonaListView.as_view(), name = 'persona-list'),
    ]
```

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('personas/', include('personas.urls'))
]
```

https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-display/#listview

Plantillas esperadas por las vistas base genéricas



Plantilla para ListView

- vim personas/templates/base.html
- vim personas/templates/personas/persona_list.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      {% block title %}
      {% endblock %}
    </title>
  </head>

  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

Listado de Personas

1 - Alfredo

2 - Juan

3 - Jorge

```
{% extends 'base.html' %}
{% block title %}
  Lista de Personas
{% endblock %}
{% block body %}
<h1> Listado de Personas </h1>
{% for instance in object_list %}
  <p>
    {{ instance.id }} -
    <a href='{{ instance.get_absolute_url }}'>{{ instance.nombres }}</a>
  </p>
{% endfor %}
{% endblock %}
```

Filtros

Un queryset es el resultado de hacer un filtro a los objetos, al estilo de un select en SQL

```
class PersonaListView(ListView):
  model = Persona
  queryset = Persona.objects.filter(edad__lte='40')
```

Listado de Personas

2 - Juan

3 - Jorge

https://docs.djangoproject.com/en/3.22/topics/db/queries/#retrieving-objects

Las opciones sobre estos filtros dependen de los:

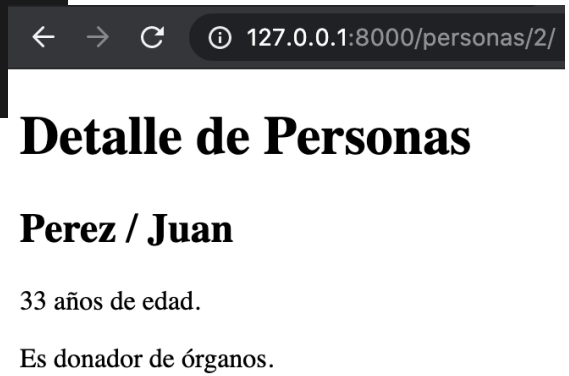
https://docs.djangoproject.com/en/3.2/topics/db/queries/#field-lookups

DetailView

```python
from django.shortcuts import render
from django.views.generic import (
    ListView,
    DetailView,
    )
from .models import Persona

# Create your views here.
class PersonaDetailView(DetailView):
  model = Persona

class PersonaListView(ListView):
  model = Persona
```

```
{% extends 'base.html' %}
{% block title %}
  Detalle de Personas
{% endblock %}
{% block body %}
<h1> Detalle de Personas </h1>
<h2> {{ object.apellidos }} / {{ object.nombres }} </h2>
<p>{{ object.edad }} años de edad.</p>
<p>
{% if object.donador %}
  Es
{% else %}
  No es
{% endif %}
donador de órganos.
</p>
{% endblock %}
```

```python
from django.urls import path
from .views import (
    PersonaListView,
    PersonaDetailView,
    )

urlpatterns = [
    path('', PersonaListView.as_view(), name = 'persona-list'),
    path('<int:pk>/', PersonaDetailView.as_view(), name = 'persona-detail')
    ]
```



https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-display/#detailview

Implementando el método get_absolute_url

```python
from django.urls import path
from .views import (
    PersonaListView,
    PersonaDetailView,
    )

app_name = 'personas'
urlpatterns = [
    path('', PersonaListView.as_view(), name = 'persona-list'),
    path('<int:pk>/', PersonaDetailView.as_view(), name = 'persona-detail')
    ]
```

```python
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:persona-detail', kwargs = {'pk': self.id})
```

view-source:127.0.0.1:8000/personas/

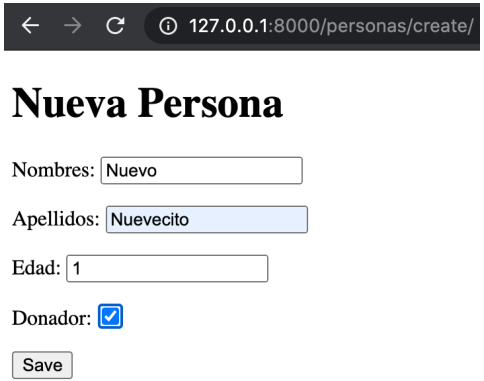
```html
<!DOCTYPE html>
<html>
  <head>
    <title>

  Lista de Personas

    </title>
  </head>

  <body>

<h1> Listado de Personas </h1>

  <p>
    1 -
    <a href='/personas/1/'>Alfredo</a>
  </p>

  <p>
    2 -
    <a href='/personas/2/'>Juan</a>
  </p>

  <p>
    3 -
    <a href='/personas/3/'>Jorge</a>
  </p>


  </body>
</html>
```

CreateView

https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-editing/#createview

vim personas/templates/personas/persona_form.html

```
{% extends 'base.html' %}
{% block title %}
  Nueva de Persona
{% endblock %}
{% block body %}
<h1> Nueva Persona </h1>
<form method="post">{% csrf_token %}
  {{ form.as_p }}
  <input type="submit" value="Save">
</form>
{% endblock %}
```




```
class PersonaCreateView(CreateView):
  model = Persona
  fields = [
      'nombres',
      'apellidos',
      'edad',
      'donador'
      ]
```

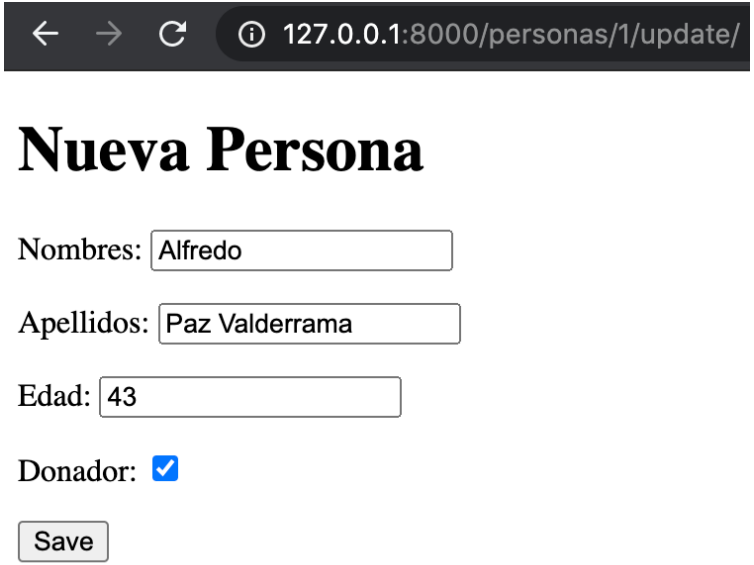
```
path('create/', PersonaCreateView.as_view(), name = 'persona-create'),
```

UpdateView

https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-editing/#updateview

- Reutiliza el formulario de CreateView
- Se puede cambiar usando:
  - template_name_suffix

```python
class PersonaUpdateView(UpdateView):
    model = Persona
    fields = [
        'nombres',
        'apellidos',
        'edad',
        'donador'
        ]
```




```python
path('<int:pk>/update/', PersonaUpdateView.as_view(), name = 'persona-update'),
```

reverse_lazy()

- Una versión perezosamente evaluada de reverse()
  - Perezosa se refiere a que no se llega a ejecutar a menos que sea necesario
- Por ejemplo, es útil para cuando necesita usar una inversión de URL antes de cargar URLConf de su proyecto.
- En nuestro caso la usaremos en la implementación de DeleteView

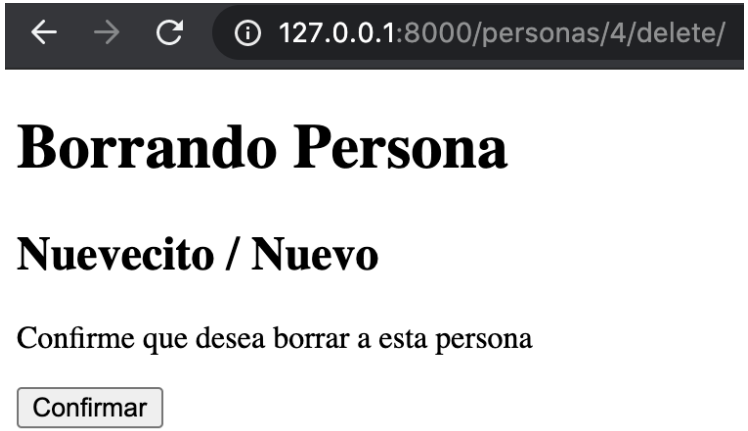
https://docs.djangoproject.com/en/3.2/ref/urlresolvers/#reverse-lazy

DeleteView

https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-editing/#deleteview

```python
from django.urls import reverse_lazy
class PersonaDeleteView(DeleteView):
  model = Persona
  success_url = reverse_lazy('personas:persona-list')
```

```
{% extends 'base.html' %}
{% block title %}
  Nueva de Persona
{% endblock %}
{% block body %}
<h1> Borrando Persona </h1>
<h2> {{object.apellidos}} / {{ object.nombres }} </h2>
<form method="post">{% csrf_token %}
  <p>
  Confirme que desea borrar a esta persona
  </p>
  <input type="submit" value="Confirmar">
</form>
{% endblock %}
```



persona_confirm_delete.html

```
path('<int:pk>/delete/', PersonaDeleteView.as_view(), name = 'persona-delete'),
```

La vista base View

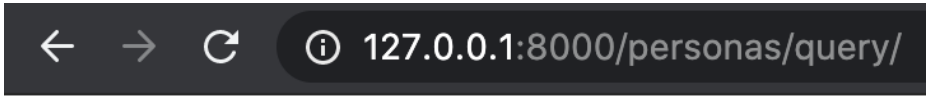
https://docs.djangoproject.com/en/3.2/ref/class-based-views/base/

La clase View es la vista base de todas las demás

A nivel de clases, las llamadas GET y POST se reciben en métodos.

```python
from django.views import View
from django.http import HttpResponse

class PersonaQueryView(View):
  def get(self, request, *args, **kwargs):
    return HttpResponse('Hola Mundo con Clases')
```

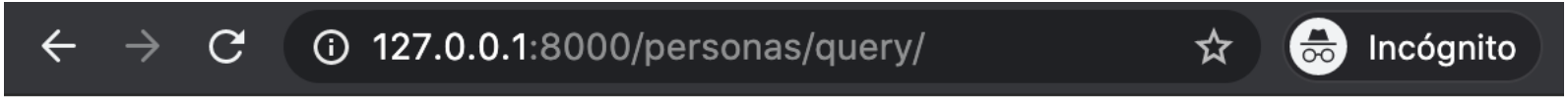
```python
path('query/', PersonaQueryView.as_view(), name = 'persona-query'),
```



Hola Mundo con Clases

Devolviendo Json

```python
class PersonaQueryView(View):
  def get(self, request, *args, **kwargs):
    queryset = Persona.objects.filter(edad__lte='40')
    return JsonResponse(list(queryset.values()), safe = False)
```




```
[{"id": 2, "nombres": "Juan", "apellidos": "Perez", "edad": 33, "donador":
true}, {"id": 3, "nombres": "Jorge", "apellidos": "Gonzales", "edad": 18,
"donador": true}]
```


https://docs.djangoproject.com/en/3.2/ref/request-response/#jsonresponse-objects

Archivos estáticos

https://docs.djangoproject.com/en/3.2/howto/static-files/

- Los sitios web generalmente necesitan servir archivos adicionales como imágenes, JavaScript o CSS. En Django, nos referimos a estos archivos como "archivos estáticos".
  - mkdir -p personas/static/personas

```
{% load static %}
<!DOCTYPE html>
<html>
  <head>
    <title>
      {% block title %}
      {% endblock %}
    </title>
  </head>

  <body>
    <img src="{% static "personas/logo-unsa.png" %}" alt="logo unsa">
    {% block body %}
    {% endblock %}
  </body>
</html>
```