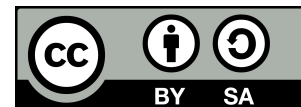


CURSO PYTHON

APLICACIONES WEB CON DJANGO



Autor: Jon Vadillo
www.jonvadillo.com



Contenidos

1. Introducción y fundamentos básicos
2. Crea tu primer proyecto en Django
3. Crea tu primera aplicación en django
4. El modelo en Django, acceso a datos y la aplicación de administrador
5. Vistas y plantillas en Django
6. **Vistas basadas en clases (Class Based Views)**
7. Formularios en Django

6. Vistas basadas en clases

Vistas mediante funciones

```
from django.http import HttpResponseRedirect

def my_view(request):
    if request.method == 'GET':
        # <view logic>
        return HttpResponseRedirect('Hola Mundo!')
```

Vistas basadas en clases

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

class MyView(View):
    saludo = 'Hola mundo!' # Atributo de la clase
    def get(self, request):
        # <view logic>
        return HttpResponse(saludo)
```

Vistas basadas en clases

- Las Vistas basadas en clases (*Class-based views*) son una alternativa para **definir vistas mediante objetos** en lugar de funciones.
- Tienen algunas ventajas como:
 - a. **Reutilización** de código y herencia
 - b. **Organización** de métodos en función del tipo de petición
- Existen **vistas genéricas basadas en clases** que ya implementan de serie soluciones a las situaciones más comunes (formularios, listas, etc.).

¿Cómo se instancian?

- Las Vistas basadas en clases contienen el método `as_view()` que devuelve una **función que será llamada por el mapeador de URLs**. En concreto realiza lo siguiente:
 1. Crea una instancia de la clase
 2. Llama al método `setup()` para inicializar los atributos
 3. En función del tipo de petición (GET/POST/etc) invoca el método oportuno.

```
# urls.py
from django.urls import path
from myapp.views import MyView

urlpatterns = [
    path('about/', MyView.as_view()),
]
```


Hands on!

Actualiza `urls.py` y `views.py` para que la aplicación utilice vistas basadas en clases en lugar de funciones.



Vistas genéricas basadas en clases

Vistas que **incluyen de serie las funcionalidades** típicamente utilizadas en todas las aplicaciones web.

DetailView

```
class EmpleadoDetailView(DetailView):  
    model = Empleado
```

```
def empleado_detail_view(request):  
    empleado = Empleado.objects.get(pk=pk)  
    return render(  
        request,  
        'empleado_detail.html',  
        {'empleado': empleado}  
    )
```

ListView

```
class EmpleadoListView(ListView):  
    model = Empleado  
    queryset = Empleado.objects.all()
```

```
def empleado_list_view(request):  
    empleado_list = Empleado.objects.all()  
    return render(  
        request,  
        'empleado_list.html',  
        {'empleados': empleado_list}  
    )
```

ListView

```
class EmpleadoListView(ListView):  
    model = Empleado  
    queryset = Departamento.objects.all()
```

```
{% extends "base.html" %}  
  
{% block content %}  
    <h2>Empleados</h2>  
    <ul>  
        {% for empleado in object_list %}  
            <li>{{ empleado.nombre }}</li>  
        {% endfor %}  
    </ul>  
{% endblock %}
```

Sobrescribir atributos

```
class EmpleadoListView(ListView):  
    model = Empleado  
    # Cambia el orden de los elementos  
    queryset = Empleado.objects.order_by('-nombre')  
    # Cambiar el nombre genérico object_list por algo más amigable  
    context_object_name = 'empleado_list'  
    # Cambiar el nombre de la plantilla utilizada  
    template_name = 'trabajador_detail.html'
```

Añadir datos a context_data

```
from django.views.generic import DetailView
from app.models import Departamento, Empleado

class EmpleadoDetail(DetailView):

    model = Empleado

    def get_context_data(self, **kwargs):
        # Cargar el contexto base
        context = super().get_context_data(**kwargs)
        # Añadir un listado de departamentos
        context['departamento_list'] = Departamento.objects.all()
        return context
```

AUTH VIEWS

LoginView
LogoutView
PasswordChangeDoneView
PasswordChangeView
PasswordResetCompleteView
PasswordResetConfirmView
PasswordResetDoneView
PasswordResetView

GENERIC DETAIL

DetailView

GENERIC EDIT

CreateView
DeleteView
FormView
UpdateView

GENERIC BASE

RedirectView
TemplateView
View

GENERIC LIST

ListView

GENERIC DATES

ArchiveIndexView
DateDetailView
DayArchiveView
MonthArchiveView
TodayArchiveView
WeekArchiveView
YearArchiveView

<https://ccbv.co.uk/>

Hands on!

Optimiza el código del archivo `views.py` para que la aplicación utilice vistas **genéricas** basadas en clases.



Herencia múltiple (mixins)

- La herencia múltiple (mixins) se utiliza cuando queremos **dotar a una clase con múltiples funcionalidades** (o cuando queremos **llevar una funcionalidad a muchas clases**).
- La principal limitación es que **solo una de las clases heredadas puede ser descendiente de la clase View** (p.ej.: no es posible heredar de ListView y FormView)

Mixins (JSONResponseMixin)

```
class JSONDetailView(JSONResponseMixin, BaseDetailView):  
    def render_to_response(self, context, **response_kwargs):  
        return self.render_to_json_response(  
            context,  
            **response_kwargs  
        )
```

Sources

- Documentación oficial: <https://www.djangoproject.com>
- Mozilla MDN Web Docs: <https://developer.mozilla.org>
- CCBV: <https://ccbv.co.uk/>