

Vistas basadas en Clases

Antonio Espín Herranz

Contenidos

- Introducción
- Funciones vs Clases
- Clases proporcionadas por django
 - View, TemplateView, RedirectView
- Vistas genéricas en URLConfs
- Vistas genéricas de visualización
 - DetailView, ListView
- Vistas genéricas de edición
 - FormView, CreateView, UpdateView, DeleteView

Introducción

- Las vistas genéricas están pensadas para simplificar el código de las vistas.
- Las tareas de una vista genérica:
 - Redirigir a otra página.
 - Renderizar una plantilla.
 - Mostrar páginas de listado y detalle (para un objeto).
 - Permitir a los usuarios realizar operaciones de crear, actualizar y borrar objetos.
- **API: Vistas basadas en clases:**
 - <https://docs.djangoproject.com/en/4.2/ref/class-based-views/>

Funciones vs Clases

- Organizan el código en **métodos HTTP** (GET, POST, ...)
 - Dentro de las funciones teníamos que estar preguntando por el método Http.
- Utilizan la **herencia múltiple** (mixins) para factorizar el código en componentes comunes y reutilizables.

Funciones vs Clases

- Con Funciones:

```
from django.http import  
    HttpResponse
```

```
def mi_vista(request):  
    if request.method == 'GET':  
        # <la lógica de la vista>  
        return  
        HttpResponse('resultado')
```

Las vistas basadas en clases tienen el método:
as_view()
Sirve para enlazar la URL con la clase

- Con Clases:

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

```
class MiVista(View):  
    def get(self, request):  
        # <la lógica de la vista>  
        return  
        HttpResponse('resultado')
```

En urls.py

- Para enlazar clase con la URL.
from django.conf.urls import url
from myapp.views import MiVista
urlpatterns = [
 path(r'^indice/', **MiVista.as_view()**),
]

Vista Base

- Django proporciona las clases:
 - **View:**
 - Maneja las conexiones de las vistas y las URLs a través de los métodos HTTP.
 - **TemplateView:**
 - Para el renderizado de plantillas
 - **RedirectView:**
 - Para redireccionamiento Http
- El resto de funciones, se consigue con **mixins** (herencia múltiple)

- **Clase base maestra.**
 - **Todas** las vistas **heredan de esta**
 - Se encuentra en: **django.views.generic.base.View**

- **views.py:**

```
from django.http import HttpResponse
from django.views.generic import View
class MiVista(View):
    def get(self, request, *args, **kwargs):
        return HttpResponse('Hola, Mundo')
```

- **urls.py:**

```
from django.conf.urls import url
from myapp.views import MiVista
```

```
urlpatterns = [
    path(r'^hola/$', MiVista.as_view(), name='mivista'),
]
```

View acepta todos
Los métodos Http:

get,
post,
put
head

...

Sobrescribimos el método

View

- **Métodos:**

- **dispatch()** que **delega la petición en el método adecuado.**
 - Una petición GET se delega en el método get()
 - POST al método post()
 - Así sucesivamente
- Si se llama a la vista con un método NO soportado se llama al método:
 - **http_method_not_allowed()** que devuelve **HttpResponseNotAllowed**
- **options()**
 - Manejadores que responden a la petición **OPTIONS HTTP**

- **Atributos:**

- **http_methods_name:**
 - Una lista con los nombres de los métodos http
 - ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']

TemplateView

- Renderiza una plantilla dada, con el contexto que contiene los parámetros capturados en la URL.
- **Métodos:**
 - **dispatch():**
 - Valida la petición (idem de View)
 - **http_method_not_allowed():**
 - Verifica los métodos soportados.
 - **get_context_data():**
 - Se encarga de pasarle el contexto (context) a la vista.
- **Atributos:**
 - **template_name:** El nombre de la plantilla.

Las plantillas

- Django sigue la siguiente convención para los nombres de las plantillas.
 - nombre_app/templates/nombre_app/
- Dentro de la carpeta de la aplicación, se crea la carpeta templates y dentro se vuelve a repetir el nombre de la app.
 - Esto lo hace así por si tenemos más de una aplicación y sepa encontrar la plantilla que corresponde a esa aplicación.

Ejemplo

- **views.py:**

```
from django.views.generic.base import TemplateView
from articles.models import Article
```

```
class HomePageView(TemplateView):
```

```
    # Sobrescribe el att.
```

```
    template_name = "home.html"
```

```
    # Sobrescribe el método
```

```
    def get_context_data(self, **kwargs):
```

```
        # Llama al método de la clase Padre.
```

```
        context = super().get_context_data(**kwargs)
```

```
        # Añade contenido al contexto, para ello accede al modelo.
```

```
        context['latest_articles'] = Article.objects.all()[:5]
```

```
        return context
```

- **urls.py**

```
from django.urls import path
```

```
from myapp.views import HomePageView
```

```
urlpatterns = [
```

```
    path("", HomePageView.as_view(), name='home'),
```

```
]
```

RedirectView

- La clase **RedirectView** redirecciona una vista con la URL dada.
- **Métodos:**
 - **dispatch()**
 - **http_method_not_allowed()**
 - **get_redirect_url():**
 - Construye el URL del objetivo para el redireccionamiento.
 - La implementación por defecto usa la url como la cadena de inicio para realizar la expansión mediante el marcador de posición % en la cadena usando el grupo de nombres capturados en la URL.

RedirectView

- **Atributos**

- **query_string:**

- boolean, por defecto False
 - Cuando es True si hay algún parámetro en la querystring se pasará a la página redirigida.

- **url:**

- La URL para redireccionar la vista. Si es None lanzará el error HTTP 410

- **permanent:** True, si el redireccionamiento debe ser permanente.

- **pattern_name:** El nombre del patrón URL para redireccionar la vista.

Ejemplo: Views

```
from django.shortcuts import get_object_or_404
from django.views.generic.base import RedirectView
from articles.models import Article
```

```
class ArticleCounterRedirectView(RedirectView):
    permanent = False
    query_string = True
    pattern_name = 'article-detail'

    def get_redirect_url(self, *args, **kwargs):
        article = get_object_or_404(Article, pk=kwargs['pk'])
        article.update_counter()
        return super().get_redirect_url(*args, **kwargs)
```

Ejemplo: urls

```
from django.urls import path  
from django.views.generic.base import RedirectView  
from article.views import ArticleCounterRedirectView,  
ArticleDetail  
  
urlpatterns = [  
    path('counter/<int:pk>/', ArticleCounterRedirectView.as_view(),  
        name='article-counter'),  
    path('details/<int:pk>/', ArticleDetail.as_view(), name='article-detail'),  
    path('go-to-django/',  
        RedirectView.as_view(url='https://djangoproject.com'), name='go-to-  
django'),  
]
```


Vistas basadas en Clases en URLconfs

- Las **vistas genéricas** basadas en clases también **se pueden utilizar directamente en el fichero urls.py** y nos ahorramos escribir la clase.
- En este ejemplo, una clase que heredara de **TemplateView**

```
from django.conf.urls import url
```

```
from django.views.generic import TemplateView
```

```
urlpatterns = [ # En el mismo método se pasa la plantilla  
    path(r'^acerca/',  
        TemplateView.as_view(template_name="acerca_de.html")),  
]
```

Vistas genéricas de visualización

- <https://docs.djangoproject.com/en/4.2/ref/class-based-views/generic-display/>
- Estas vistas tienen que ver con el modelo para mostrar el **detalle de un objeto (DetailView)** o una **lista de objetos (ListView)**.
- **Django** puede **inferir** los nombres de las **plantillas** utilizando el modelo y el tipo de vista.
- **Por ejemplo:**
 - Si el modelo es **Editor** y utilizamos la vista **DetailView**, la plantilla a inferir será → **editor_detail.html**
 - Con **Autor** y **ListView** será → **autor_list.html**


Ejemplo (parte 1)

Petición:

<http://localhost:8000/editores>

URLs

```
from django.conf.urls import path
from biblioteca.views import ListaEditores
urlpatterns = [
    path(r'^editores/$', ListaEditores.as_view(), name='listaeditores'),]
```



• MODELO:

- from django.db import models
- class **Editor**(models.Model):
 - nombre = models.CharField(max_length=30)
 - domicilio = models.CharField(max_length=50)
 - ciudad = models.CharField(max_length=60)
 - estado = models.CharField(max_length=30)
 - pais = models.CharField(max_length=50)
 - website = models.URLField()
- def __str__(self):
 - return self.nombre

VIEWS

```
from django.views.generic import ListView
from biblioteca.models import Editor
class ListaEditores(ListView):
    model = Editor
```

Ejemplo (parte 2)

- **PLANTILLA: editor_list.html (inferida)**

```
{% extends "base.html" %}
{% block content %}
<h2>Editores</h2>
{% if object_list %}
<ul>
    {% for editores in object_list %}
    <li><a href="{% url 'detalleseditor' editores.pk %}">
        {{ editores.nombre }}</a></li>
    {% endfor %}
</ul>
{% else %}
    <p>No hay editores registrados.</p>
{% endif %}
{% endblock %}
```

URLs

```
from django.conf.urls import path
from biblioteca.views import DetallesEditor
urlpatterns = [
    path(r'^detalles/editor/(?P<pk>[09]+)/$', DetallesEditor.as_view(),
         name='detalleseditor'
    ), ]
```

VIEWS

```
from django.views.generic.detail import DetailView
from biblioteca.models import Editor
class DetallesEditor(DetailView):
    model = Editor
```

Ejemplo (parte 3)

- **PLANTILLA: editor_detail.html**

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

```
{% block content %}
```

```
    <h2>Editor: {{ editor.nombre }}</h2>
```

```
    <ul>
```

```
        <li>Domicilio: {{ editor.domicilio }}</li>
```

```
        <li>Ciudad: {{ editor.ciudad }}</li>
```

```
        <li>Estado: {{ editor.estado }}</li>
```

```
        <li>Pais: {{ editor.pais }}</li>
```

```
        <li>Sitio web: {{ editor.website }}</li>
```

```
    </ul>
```

```
    <p><a href="{% url 'listaeditores' %}">Lista de editores</a></p>
```

```
{% endblock %}
```

Notas

- El nombre de la plantilla se infiere con el nombre del modelo + '___' + el tipo de vista detail / list. Con todas las letras en minúsculas.
- Dentro de la vista Detail el objeto se llama igual que el modelo pero en minúsculas.
 - Editor → **editor**
- Dentro de la plantilla asociada a la vista ListView la lista de objetos se llama:
 - **objectlist**

Vistas genéricas de Edición

- Todas estas vistas se encuentran en el paquete:
 - `django.views.generic.edit`.
 - <https://docs.djangoproject.com/en/4.2/ref/class-based-views/generic-editing/>
- **FormView:**
 - **Una vista que muestra un formulario. EN PRINCIPIO NO INTERACTUA CON LA BD (los otros 3 si)**
 - En caso de error, vuelve a mostrar el formulario con errores de validación;
 - En caso de éxito, redirige a una nueva URL.
- **CreateView**
 - **Una vista que muestra un formulario para crear un objeto**, volver a mostrar el formulario con errores de validación (si existen) y guardar el objeto.
- **UpdateView**
 - **Una vista que muestra un formulario para editar un objeto existente**, volver a mostrar el formulario con errores de validación (si existen) y guardar los cambios en el objeto.
 - Este utiliza un formulario generado automáticamente a partir de la clase modelo del objeto (a menos que se especifique manualmente una clase de formulario).
- **DeleteView**
 - **Una vista que muestra una página de confirmación y elimina un objeto existente**. El objeto dado solo se eliminará si el método de solicitud es POST. Si se obtiene esta vista a través de GET, se mostrará una página de confirmación que debe contener un formulario que se envía a la misma URL.

Ejemplos

- **Modelo de prueba:**

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

```
class Author(models.Model):
    name = models.CharField(max_length=200)
    def get_absolute_url(self):
        return reverse('author-detail', kwargs={'pk': self.pk})
```

- author-detail es el nombre de una url, atributo name.
- reverse recupera la url a partir del nombre 'author-detail'

FormView

- Con el **form**:

```
from django import forms  
class ContactForm(forms.Form):  
    name = forms.CharField()  
    message = forms.CharField(widget=forms.Textarea)  
    def send_email(self):  
        pass
```

```
from myapp.forms import ContactForm  
from django.views.generic.edit import FormView
```

```
class ContactView(FormView):  
    template_name = 'contact.html'  
    form_class = ContactForm  
    success_url = '/thanks/'
```

```
def form_valid(self, form):
```

*# Este método se llama cuando los datos del formulario son válidos. Y tiene
retornar un objeto HttpResponse, para esto se apoya en la clase Padre.*

```
form.send_email()
```

```
return super().form_valid(form)
```

LA PLANTILLA: contact.html

```
<form method="post">{% csrf_token %}  
{{ form.as_p }}  
<input type="submit" value="Send message">  
</form>
```

CreateView

- **from django.views.generic.edit import CreateView**
- **from myapp.models import Author**
- **class AuthorCreate(CreateView):**
 - **model = Author** *# La clase del modelo*
 - **fields = ['name']** *# Los campos del form*
- La plantilla inferida: **author_form.html**
 - **<form method="post">{% csrf_token %}**
 - **{{ form.as_p }}**
 - **<input type="submit" value="Save">**
 - **</form>**

UpdateView

```
from django.views.generic.edit import UpdateView
```

```
from myapp.models import Author
```

```
class AuthorUpdate(UpdateView):
```

```
    model = Author
```

```
    fields = ['name']
```

```
    template_name_suffix = '_update_form'
```

- La **plantilla** se infiere con el **nombre del modelo** + el att. **template_name_suffix**
- **Plantilla: author_update_form.html**
 - `<form method="post">{% csrf_token %}`
 - `{{ form.as_p }}`
 - `<input type="submit" value="Update">`
 - `</form>`

DeleteView

```
from django.urls import reverse_lazy
from django.views.generic.edit import DeleteView
from myapp.models import Author
```

```
class AuthorDelete(DeleteView):
    model = Author
    fields = ['name']
    success_url = reverse_lazy('author-list')
```

- **La plantilla: author_confirm_delete.html**

```
<form method="post">{% csrf_token %}
    <p>Are you sure you want to delete "{{ object }}"?</p> <input type="submit"
        value="Confirm">
</form>
```

reverse_lazy: Lo mismo que reverse pero puede que no se haya cargado la url