Django - Vistas Basadas en Clases

Metodología de la Programación

UNJu – Facultad de Ingeniería

Ing. José Zapana

Introducción

- Todas las vistas que hemos creado en nuestros proyectos han sido vistas basadas en funciones **FBV**.
- Se definen en simples funciones que luego llamamos en el respectivo fichero urls.py.
- Estas eran las únicas vistas que se podían crear hasta que llegaron las vistas basadas en clases **CBV**

Vistas Basadas en Clases (CBV)

Django dispone de un amplio catálogo de vistas basadas en clases a modo de plantillas que podemos utilizar y extender.

Ventajas:

- Una clase es una estructura mucho más completa que una simple función. Permite usarla de molde y puede contener comportamientos predefinidos que se pueden heredar en otras clases
- Tienen un objetivo muy claro: ahorrarnos tiempos a la hora de implementar funcionalidades comunes.

La documentación detallada puede verse en:
https://ccbv.co.uk/

¿Cómo utilizarlas?

- Hay que seleccionar el CBV según nuestras necesidades
- Implementarla en nuestra aplicación en base a la documentación correspondiente
- Por ejemplo:
  - TemplateView
  - ListView
  - CBV CRUD CreateView
  - CBV CRUD UpdateView
  - CBV CRUD DeleteView
  - otros

TemplateView

Se utiliza para páginas que gestionan contenido básico

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

from articles.models import Article

class HomePageView(TemplateView):

    template_name = "home.html"

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['latest_articles'] = Article.objects.all()[:5]
        return context
```

```python
from django.urls import path

from myapp.views import HomePageView

urlpatterns = [
    path('', HomePageView.as_view(), name='home'),
]
```

TemplateView - Ejemplo

Enviar información mediante el diccionario de contexto del template.

```python
from django.views.generic.base import TemplateView
from django.shortcuts import render

class HomePageView(TemplateView):
    template_name = "core/home.html"

    def get(self, request, *args, **kwargs):
        return render(request, self.template_name, {'title':"MP Inicio"})
```

```
<h1>{{title}}</h1>
```

ListView

Se utiliza para devolver una lista de objetos de un modelo determinado

[class ListView]

[class django.views.generic.list.ListView]

```python
from django.views.generic.list import ListView
from django.utils import timezone

from articles.models import Article

class ArticleListView(ListView):

    model = Article

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['now'] = timezone.now()
        return context
```

Example myapp/article_list.html:

```
<h1>Articles</h1>
<ul>
{% for article in object_list %}
    <li>{{ article.pub_date|date }} - {{ article.headline }}</li>
{% empty %}
    <li>No articles yet.</li>
{% endfor %}
</ul>
```

ListView - Ejemplo

Implementar el listado de instrumentos

```python
# Create your views here.
class GuitarraListView(ListView):
    model = Guitarra
```

```python
urlpatterns = [
    path('', views.home),
    path('about-me', GuitarraListView.as_view(),name='guitarras'),
```


```
{% for guitarra in guitarra_list %}
    <div class="row ">
        <div class="col-lg-3 col-md-4 offset-lg-1">
            <img class="img-fluid" src="{{guitarra.imagen.url}}" alt="">
        </div>
        <div class="col-lg-7 col-md-8">
```

CreateView

Permite dar de alta a un nuevo objeto

[class CreateView]

[class django.views.generic.edit.CreateView]

```python
from django.views.generic.edit import CreateView
from myapp.models import Author

class AuthorCreate(CreateView):
    model = Author
    fields = ['name']
```

Example myapp/author_form.html:

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

CreateView - Ejemplo

```python
class GuitarraCreate(CreateView):
    model = Guitarra
    #fields = ['nombre', 'descripcion', 'imagen']
    success_url = reverse_lazy('guitarras')
```

```python
urlpatterns = [
    path('', views.home),
    path('about-me', GuitarraListView.as_view(),name='guitarras'),
    path('create/', GuitarraCreate.as_view(), name='create'),
```

```html
<form action="" method="post"  enctype="multipart/form-data">{% csrf_token %}
    {{ form.as_p }}
  <div class="text-center">
    <input type="submit" class="btn btn-primary btn-block" value="Crear" />
  </div>
</form>
```

guitarra_form.html

UpdateView

Permite modificar un objeto

- class UpdateView
- class django.views.generic.edit.UpdateView

```python
from django.views.generic.edit import UpdateView
from myapp.models import Author

class AuthorUpdateView(UpdateView):
    model = Author
    fields = ['name']
    template_name_suffix = '_update_form'
```

Example myapp/author_update_form.html:

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

UpdateView - Ejemplo

```python
class GuitarraUpdate(UpdateView):
    model = Guitarra
    #fields = ['nombre', 'descripcion', 'imagen']
    form_class = GuitarraForm
    template_name_suffix = '_update_form'

    def get_success_url(self):
        return reverse_lazy('guitarras')+'?Actualizado'
```

```python
urlpatterns = [
    path('', views.home),
    path('about-me', GuitarraListView.as_view(),name='guitarras'),
    path('create/', GuitarraCreate.as_view(), name='create'),
    path('update/<int:pk>/', GuitarraUpdate.as_view(), name='update'),
```

```html
<form action="" method="post"  enctype="multipart/form-data">{% csrf_token %}
    {{ form.as_p }}
  <div class="text-center">
    <input type="submit" class="btn btn-primary btn-block" value="Actualizar" />
  </div>
</form>
```

guitarra_update_form.html

DeleteView

Permite eliminar un objeto

- class DeleteView
- class django.views.generic.edit.DeleteView

Example myapp/views.py:

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

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

Example myapp/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>
```

DeleteView - Ejemplo

```python
class GuitarraDelete(DeleteView):
    model = Guitarra
    def get_success_url(self):
        return reverse_lazy('guitarras')+'?Eliminado'
```

```python
urlpatterns = [
    path('', views.home),
    path('about-me', GuitarraListView.as_view(),name='guitarras'),
    path('create/', GuitarraCreate.as_view(), name='create'),
    path('update/<int:pk>/', GuitarraUpdate.as_view(), name='update'),
    path('delete/<int:pk>/', GuitarraDelete.as_view(), name='delete'),
```

```html
<form action="" method="post">{% csrf_token %}
  <p>¿Estás seguro de que quieres borrar <b>"{{ object }}"</b>?</p>
  <input type="submit" class="btn btn-primary btn-block" value="Confirmar" />
  <a class="nav-link" href="{% url 'guitarras' %}">Cancelar</a>
</form>
```

guitarra_confirm_delete.html

Temas de estudio futuro

- Validación y autenticación (LoginForm)
  - Decoradores
- Paginación
- Crear una API
- Despliegue de la aplicación:
  - Configuración del entorno de producción
  - https://www.pythonanywhere.com/

Referencias

- https://ccbv.co.uk/