Django Class-Based Views (CBVs): An Overview and Practical Example

Nuno Bispo

Class-Based Views (CBVs), allow developers to handle HTTP methods, such as GET and POST, with class instances instead of functions.

Image by Author

Django, a high-level Python web framework, encourages rapid development and clean, pragmatic design.

One of its powerful features is the Class-Based Views (CBVs), which allow developers to handle HTTP methods, such as GET and POST, with class instances instead of functions.

This article provides an overview of CBVs and demonstrates their use with a simple application.

Complete source code available at:

GitHub - nunombispo/django-cbv: Django Class Based Views

Django Class Based Views. Contribute to nunombispo/django-cbv development by creating an account on GitHub.

Blog Application

Class-Based Views and its Advantages:

A Class-Based View is essentially a Python class that inherits from one or more of Django's provided view classes.

The most basic CBV, `View`, handles different HTTP methods with methods named after them (e.g., `get()`, `post()`).

Reusability

CBVs promote the reuse of common view logic through class inheritance.

Modularity

CBVs can be extended with mixins, allowing developers to add functionality without modifying the base view.

Clarity

CBVs can make the code more organized, especially when dealing with views that have multiple HTTP methods.

Practical Example: A Simple Blog Application

Let's create a basic blog application using CBVs.

We start by creating a new app called 'blog':

```
django-admin startapp blog
```

Then we add it to the INSTALLED_APPS settings:

```
INSTALLED_APPS = [
    ...
    'blog'
]
```

Let's now build our app logic, starting with the models.

models.py:

```
from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_ad
```

And we run the migrations for the new model:

```
python manage.py makemigrations
python manage.py migrate
```

urls.py:

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

urlpatterns = [
    path('', views.BlogListView.as_view(), name=
    path('<int:pk>/', views.BlogDetailView.as_vie
]
```

Don’t forget to add the Blog app URLs to the project’s main URLs:

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

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

views.py:

```
from django.views.generic import ListView, Detai
from .models import Blog

class BlogListView(ListView):
    model = Blog
    template_name = 'blog_list.html'
    context_object_name = 'blogs'
class BlogDetailView(DetailView):
    model = Blog
    template_name = 'blog_detail.html'
    context_object_name = 'blog'
```

blog_list.html:

```
{% for blog in blogs %}
    <h2><a href="{% url 'blog_detail' blog.pk %}'
{% endfor %}
```

blog_detail.html:

```
<h1>{{ blog.title }}</h1>
<p>{{ blog.content }}</p>
```

Explanation:

BlogListView: This CBV displays a list of all blog posts. It uses Django's `ListView`, which automatically fetches the queryset and passes it to the template.

The `context_object_name` attribute is used to specify the name of the variable in the template.

BlogDetailView: This CBV displays a single blog post based on its primary key.

It uses Django's `DetailView` which fetches the object based on the primary key provided in the URL and passes it to the template.

Creating and Deleting with CBVs

Let's extend our simple blog application to include functionality for creating and deleting blog posts using Class-Based Views.

Creating a Blog Post

To create a blog post, we'll use Django's `CreateView`.

forms.py:

```
from django import forms
from .models import Blog

class BlogForm(forms.ModelForm):
    class Meta:
        model = Blog
        fields = ['title', 'content']
```

views.py (addition):

```
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from .models import Blog
from .forms import BlogForm

class BlogCreateView(CreateView):
    model = Blog
    form_class = BlogForm
    template_name = 'blog_create.html'
    success_url = reverse_lazy('blog_list')
```

blog_create.html:

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Create Blog</button>
</form>
```

blog_list.html (addition):

```
<a href="{% url 'blog_create' %}">Add new blog</a
```

urls.py (addition):

```
path('create/', views.BlogCreateView.as_view(), n
```

Deleting a Blog Post

To delete a blog post, we'll use Django's `DeleteView`.

views.py (addition):

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

class BlogDeleteView(DeleteView):
    model = Blog
    template_name = 'blog_delete.html'
    success_url = reverse_lazy('blog_list')
```

blog_delete.html:

```
<form method="post">
    {% csrf_token %}
    <p>Are you sure you want to delete "{{ object
    <button type="submit">Yes, Delete</button>
    <a href="{% url 'blog_list' %}">Cancel</a>
</form>
```

blog_detail.html (addition):

```
<a href="{% url 'blog_delete' blog.pk %}">DELETE<
```

urls.py (addition):

```
path('<int:pk>/delete/', views.BlogDeleteView.as_
```

Explanation:

BlogCreateView: This CBV displays a form for creating a new blog post and handles the form submission, handling both the GET request to display the form and the POST request to submit the form.

It uses Django's `CreateView`, which automatically manages the form display, validation, and saving of the object to the database.

The `success_url` attribute specifies where to redirect after a successful form submission.

BlogDeleteView: This CBV displays a confirmation page for deleting a blog post and handles the deletion, handling both the GET request to display the blog info and the and POST request to handle the record deletion.

It uses Django's `DeleteView`, which fetches the object based on the primary key provided in the URL and deletes

it upon confirmation.

Thank you for reading and I will see you on the Internet.

Follow me on Twitter: https://twitter.com/DevAsService

Check out my website at: https://developer-service.io/