```
{% comment %}
    Usage: {% include "flavors/_flavor_search.html" %}
{% endcomment %}
<form action="{% url "flavor_list" %}" method="GET">
    <input type="text" name="q" />
    <button type="submit">search</button>
</form>
```

TIP: Specify the Form Target in Search Forms

We also take care to specify the URL in the form action, because we've found that search forms are often included in several pages. This is why we prefix them with '_' and create them in such a way as to be included in other templates.

Once we get past overriding the `ListView's get_queryset()` method, the rest of this example is just a simple HTML form. We like this kind of simplicity.

9.6 Using Just django.views.generic.View

It's entirely possible to build a project just using `django.views.generic.View` for all the views. It's not as extreme as one might think. For example, if we look at the official Django documentation's introduction to class-based views (http://2scoops.co/1.6-using-cbvs), we can see the approach is very close to how function-based views are written. In fact, we highlighted this two chapters ago in subsection 7.6.1 'The Simplest Views' because it's important.

Imagine instead of writing function-based views with nested-ifs representing different HTTP methods or class-based views where the HTTP methods are hidden behind `get_context_data()` and `form_valid()` methods, they are readily accessible to developers. Imagine something like:

Example 9.13

```
from django.shortcuts import get_object_or_404
from django.shortcuts import render, redirect
from django.views.generic import View
```

```
from braces.views import LoginRequiredMixin

from .forms import FlavorForm
from .models import Flavor

class FlavorView(LoginRequiredMixin, View):

    def get(self, request, *args, **kwargs):
        # Handles display of the Flavor object
        flavor = get_object_or_404(Flavor, pk=kwargs['slug'])
        return render(request,
            "flavors/flavor_detail.html",
                {"flavor": flavor}
            )

    def post(self, request, *args, **kwargs):
        # Handles updates of the Flavor object
        flavor = get_object_or_404(Flavor, pk=kwargs['slug'])
        form = FlavorForm(request.POST)
        if form.is_valid():
            form.save()
        return redirect("flavors:detail", flavor.slug)
```

While we can do this in a function-based view, it can be argued that the GET/POST method declarations within the `FlavorView` are more easier to read than the traditional "`if request.method == ...`" conditions. In addition, since the inheritance chain is so shallow, it means using mixins doesn't threaten us with cognitive overload.

What we find really useful, even on projects which use a lot of generic class-based views, is using the `django.views.generic.View` class with a GET method for displaying JSON, PDF or other non-HTML content. All the tricks that we've used for rendering CSV, Excel, and PDF files in function-based views apply when using the GET method. For example:

Example 9.14

```
from django.http import HttpResponse
from django.shortcuts import get_object_or_404
```

```
from django.views.generic import View

from braces.views import LoginRequiredMixin

from .models import Flavor
from .reports import make_flavor_pdf

class PDFFlavorView(LoginRequiredMixin, View):

    def get(self, request, *args, **kwargs):
        # Get the flavor
        flavor = get_object_or_404(Flavor, pk=kwargs['slug'])

        # create the response
        response = HttpResponse(mimetype='application/pdf')

        # generate the PDF stream and attach to the response
        response = make_flavor_pdf(response, flavor)

        return response
```

This is a pretty straight-forward example, but if we have to leverage more mixins and deal with more custom logic, the simplicity of `django.views.generic.View` makes it much easier than the more heavyweight views. In essence, we get all the advantages of function-based views combined with the object-oriented power that CBVs gave us starting with Django 1.3.

9.7 Additional Resources

- http://2scoops.co/1.6-topics-class-based-views
- http://2scoops.co/1.6-cbv-generic-display
- http://2scoops.co/1.6-cbv-generic-editing
- http://2scoops.co/1.6-cbv-mixins
- http://2scoops.co/1.6-ref-class-based-views
- The GCBV inspector at http://ccbv.co.uk
- www.python.org/download/releases/2.3/mro/