What I have learned along the way

(or: What I wish I knew when I started)

Werner Fortmann

Macquarie University
Department of Actuarial Studies
Department of Economics
e: werner.fortmann@gmail.com

July 12, 2012

Some things for those who are just beginning.

1. **CSS**
   - Use a framework such as:
     - ★ bootstrap (`http://twitter.github.com/bootstrap/`) ; or
     - ★ foundation (`http://foundation.zurb.com/`)
   - Just google “css framework” for others.
2. **Javascript**
   - Spend some time playing around with jQuery
3. **Django Templating**
   - Learn how to use the Django templating engine. Particularly writing a base template that you then extend. For a good example of a base template have a look at how pinax (`http://pinaxproject.com/`) write theirs.
4. **Django URL dispatcher**
5. **Django Views**

Django URL dispatcher

reverse() and reverse_lazy()

These are used in your code to retrieve a particular URL from your URLConf. Keeps everything nice and consistent.

- **reverse()**
- **reverse_lazy()** (django 1.4+)
  - It is useful for when you need to use a URL reversal before your project's URLConf is loaded.

{% url %} template tag

Returns an absolute path reference (a URL without the domain name) matching a given view function and optional parameters. This is a way to output links without violating the DRY principle by having to hard-code URLs in your templates

Django URL dispatcher - Example Code

Name your URLs:

```
url(r'^(?P<pk>\d+)/view/$', SyDjangoDetail.as_view(),
      name = 'sydjango-view')
```

In your code use reverse() and reverse_lazy():

```
from django.core.urlresolvers import reverse

return HttpResponseRedirect(reverse('sydjango-view',
  kwargs={'pk':sydjango.id}))
```

In your templates use the {% url %} template tag:

```
{% if sydjango %}
  <a href="{%  url sydjango-view pk=sydjango.id %}">View</a>
{% endif %}
```

Custom Model Manager

A Manager is the interface through which database query operations are provided to Django models. Adding extra Manager methods is the preferred way to add ”table-level” functionality to your models.

```
from django.db import models
from django.db.models.query import QuerySet
import pandas

class ForturusQuerySet(QuerySet):
 def to_df(self):
  field_names = [field.name for field in self.model._meta.fields]
  qs_values = self.values_list(*field_names)
  return pandas.DataFrame(list(qs_values),
      columns = field_names)
    if len(qs_values) else pandas.DataFrame(columns = field_names)

class ForturusManager(models.Manager):
 def get_query_set(self):
  return ForturusQuerySet(self.model, using=self._db)
```

Custom Model Manager

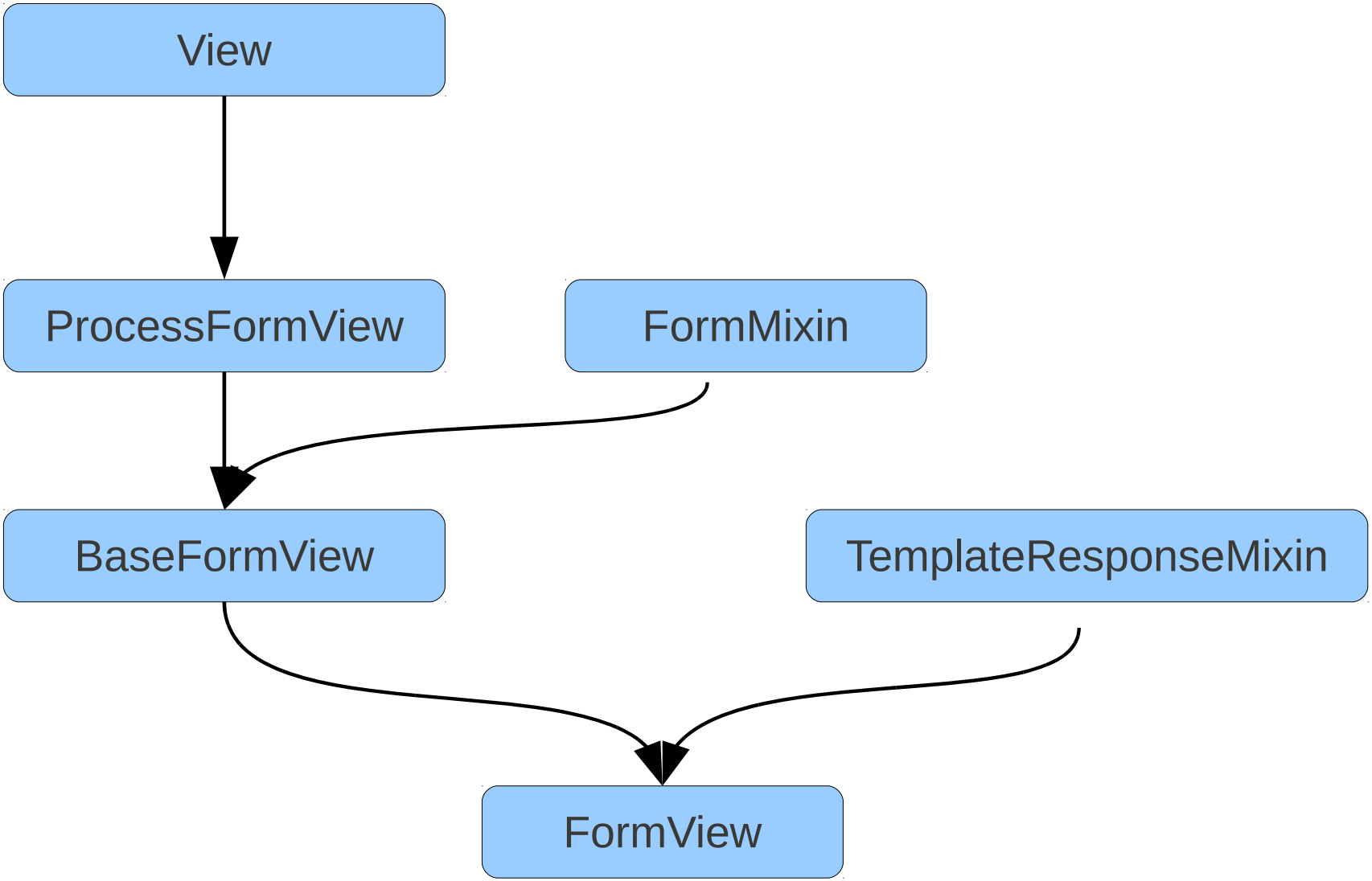
A Manager is the interface through which database query operations are provided to Django models. Adding extra Manager methods is the preferred way to add ”table-level” functionality to your models.

```
from forturus.managers import ForturusManager

class GICS(models.Model):
  id = models.AutoField(primary_key=True)
  gics_code = models.IntegerField(...)
  gics_type = models.CharField(...)
  class_name = models.CharField(...)
  class_desc = models.TextField(...)

  # Apply the custom manager.
  objects = ForturusManager()
```

FormView Inheritance



FormView Inheritance

The methods we will be dealing with for constructing the CSV upload functionality are:

View (django.views.generic.base)

- dispatch(self, request, *args, **kwargs):

ProcessFormMixin (django.views.generic.edit)

- get(self, request, *args, **kwargs):
- post(self, request, *args, **kwargs):

FormMixin (django.views.generic.edit)

- form_valid(self, form):

CSV Upload - Multi Stage Form

```
class UploadCSV_View1(FormView):

  template_name = 'uploadcsv_view1.html'
  form_class = UploadFileForm

  def form_valid(self,form):
    form = self.form_class(self.request.POST, self.request.FILES)
    if form.is_valid():
      from pandas.io.parsers import read_csv

      fh = self.request.FILES['docfile']
      csv_df = read_csv(filepath_or_buffer = fh)
      self.request.session['assettrans_csvupload_df'] = csv_df

      return HttpResponseRedirect(reverse('UploadCSV_view2',
            kwargs={'pk':self.kwargs['pk']}))
    else:
      return self.render_to_response(self.get_context_data(form=fo
```

CSV Upload - Multi Stage Form

```
class UploadCSV_View2(FormView):

    template_name = 'uploadcsv_view2.html'
    assettrans_fields = [{'field':'data_date', 'label':"Date"},
            {'field':'trans_type', 'label':"Transaction Type"}]
```

CSV Upload - Multi Stage Form

```
class UploadCSV_View2(FormView):
  def dispatch(self, request, *args, **kwargs):
    if request.method.lower() in self.http_method_names:
      handler = getattr(self, request.method.lower(), self.http_me
    else:
      handler = self.http_method_not_allowed
    self.request = request
    self.args = args
    self.kwargs = kwargs

    if not self.request.session['assettrans_csvupload_df']:
      return HttpResponseRedirect(reverse('UploadCSV_view1',
          kwargs = {'pk':self.kwargs['pk']}))
    else:
      choices = gen_choices(self.request.session['assettrans_csvup
      class formset_class(forms.Form):
        assettrans_field = forms.ChoiceField(choices = choices)
      self.form_class = formset_factory(formset_class,
          extra = len(self.assettrans_fields))
    return handler(request, *args, **kwargs)
```

CSV Upload - Multi Stage Form

```
class UploadCSV_View2(FormView):

  def post(self, request, *args, **kwargs):
    form_class = self.get_form_class()
    form = self.get_form(form_class)
    if form.is_valid():

      csv_df = self.request.session['assettrans_csvupload_df']
      field_map =
        [int(field.get('assettrans_field',0))
         for field in form.cleaned_data]

      assettrans_df = self.process_csv(csv_df, field_map)

      self.request.session['assettrans_df'] = assettrans_df
      return HttpResponseRedirect(reverse('UploadCSV_view3', kwarg
    else:
      return self.form_invalid(form)
```

CSV Upload - Multi Stage Form

```
class UploadCSV_View3(FormView):

  def post(self, request, *args, **kwargs):
    form_class = self.get_form_class()
    form = self.get_form(form_class)
    if form.is_valid():
      assettrans_df = self.request.session['assettrans_df']
      for asset_trans_inst in assettrans_df['asset_trans']:
        asset_trans_inst.save()

      #Send signal notifying of update to transactions data.
      update_pah_sig.send(sender=UploadCSV_View3,
        params_list = self.calc_params_list(assettrans_df))
      return self.form_valid(form)
    else:
      return self.form_invalid(form)
```