DEVELOPING REUSABLE DJANGO APPS

PyCon Ukraine 2010

Apps

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'cms',
    'contact',
)
```

django.contrib.auth

django.contrib.admin

“Do one thing, and do it well.”

Advantages

- Improve Quality and Flexibility
- Great Ecosystem
- Functionality for free
- Reuse over and over
- Cut down on time and effort
- Head start on new Projects

Non-Reusable Apps

- Feature creep
- Whole site is in one app
- Not going to reuse

Focus

What does this application do?

Good Focus

"Handle entries in a blog"

Bad Focus

"Handle entries in a blog, and users who post them, and their authentication, and tagging, and categorization ..."

Orthogonality

Changing a feature doesn’t affect the other one

Guidelines

- Unrelated Features
- Orthogonality
- Reuse

View Arguments

Flexible Form Handling

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

```
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponseRedirect('/thanks/')
    else:
        form = ContactForm()

    return render_to_response('contact.html', {
        'form': form,
    })
```

```
def contact(request, form_class=ContactForm):
    if request.method == 'POST':
        form = form_class(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponseRedirect('/thanks/')
    else:
        form = form_class()

    return render_to_response('contact.html', {
        'form': form,
    })
```

```
(r’contact/’,
  ‘contactform.views.contact’, {
     ‘form_class’: MyContactForm,
  })
```

Flexible Template Handling

```
def contact(request, form_class=ContactForm,
                template_name='contact.html'):
    if request.method == 'POST':
        form = form_class(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponseRedirect('/thanks/')
    else:
        form = form_class()

    return render_to_response(template_name, {
        'form': form,
    })
```

```
(r'contact/',
  'contactform.views.contact', {
     'form_class': MyContactForm,
     'template_name': 'contact/mycontactform.html',
  })
```

Post Form Processing

```
def contact(request, form_class=ContactForm,
            template_name='contact.html',
            success_url='/thanks/'):
    if request.method == 'POST':
        form = form_class(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponseRedirect(success_url)
    else:
        form = form_class()

    return render_to_response(template_name, {
        'form': form,
    })
```

```
(r’contact/’,
  ‘contactform.views.contact’, {
     ‘form_class’: MyContactForm,
     ‘template_name’: ‘contact/mycontactform.html’,
     ‘success_url’: ‘/’,
  })
```

Extra Context

```
def contact(request, ..., **extra_context={}**):
    if extra_context is None: extra_context = {}
    if request.method == 'POST':
        ...
    else:
        form = form_class()

t = template_loader.get_template(template_name)
c = RequestContext(request, {
   'form': form,
}, context_processors)

for key, value in extra_context.items():
    if callable(value):
        c[key] = value()
    else:
        c[key] = value
return HttpResponse(t.render(c))
```

```
(r’contact/’,
  ‘contactform.views.contact’, {
     ‘form_class’: MyContactForm,
     ‘template_name’: ‘contact/mycontactform.html’,
     ‘success_url’: ‘/’,
     ‘extra_context’: {‘foo’: ‘bar’},
  })
```

Flexible URL Handling

```
urlpatterns = patterns('',
    url(r'^$', 'contact.views.contact', name='contact'),
)
```

Flexible URL Handling

- Don't hardcode URLs

- django.db.models.permalink
- {% url %}
- django.core.urlresolvers.reverse()

```
def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render_to_response(‘post_detail.html’, {
        ‘post’: post
    })
```

```
(r’post/(\d+)/’,
  ‘django.views.generic.list_detail.object_detail’, {
     ‘queryset’: Post.objects.all()
  })
```

Generic Views

```
def object_detail(request, year, month, day, queryset, date_field,
        month_format='%b', day_format='%d', object_id=None, slug=None,
        slug_field='slug', template_name=None, template_name_field=None,
        template_loader=loader, extra_context=None, context_processors=None,
        template_object_name='object', mimetype=None, allow_future=False):
```

```
def post_detail(request, thread, slug):
  post = get_object_or_404(Post,
      thread__pk=thread,
      slug=slug
  )
  return render_to_response(‘post_detail.html’, {
    ‘post’: post
  })
```

newforms-admin

```
class BookAdmin(admin.ModelAdmin):
  def save_model(self, request, obj, form, change):
    if not change: obj.author = request.user
    obj.save()

  def has_change_permission(self, request, obj=None):
    if not obj:
      return True
    if obj.author == request.user:
      return True
    else:
      return False

admin.site.register(Book, BookAdmin)
```

Class Based Views

```
class PostView(DetailView):
    queryset = Post.objects.all()
```

```
(r’post/(?P<pk>\d+)/’, PostView.as_view())
```

```
(r’post/(?P<pk>\d+)/’, PostView.as_view(queryset=qs))
```

```
class PostView(DetailView):
  queryset = Post.objects.all()

  def get_object(thread, slug):
    return get_object_or_404(Post,
        thread__pk=thread,
        slug=slug
    )
```

Mixins

```
from jingo import render_to_string, env

class JinjaMixin(object):

  def render(self, names=None, context=None):
    template = env.select_template(names)
    return render_to_string(self.request,
                            template, context)
```

```python
class PostView(JinjaMixin, DetailView):
    queryset = Post.objects.all()
```

App objects

models.py

```
APP_CLASSES = (
    'myauth.MyAuthApp',
)
```

apps.py

```
from django.contrib.auth import AuthApp

class MyAuthApp(AuthApp):
    verbose_name = 'My Custom Authentication App'
```

Extending Models

User.get_profile()

```
from django.contrib.auth import AuthApp

class MyAuthApp(AuthApp):

    def models(self):
        return self.models.iteritems()
    models = property(models)
```

Questions?

@arthurk