Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

Admin Modules

Admin Modules: Table of content

Django Admin Basics
(things you should already know)

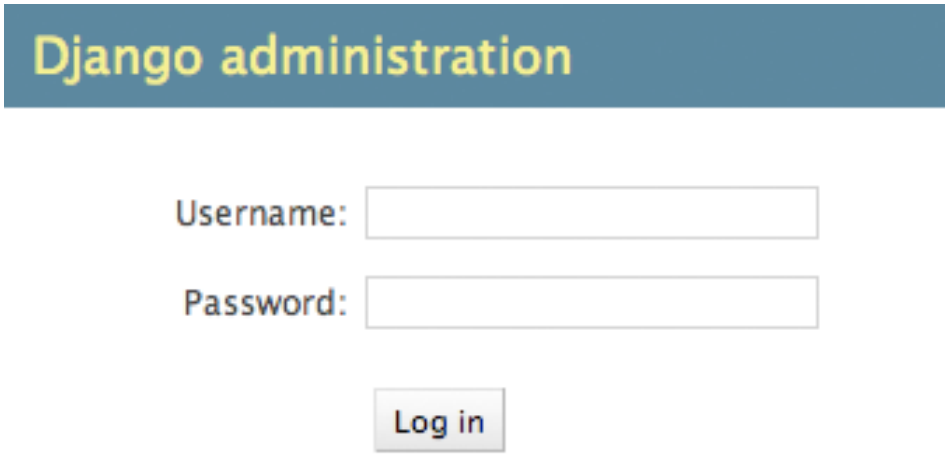
Activating the the admin site

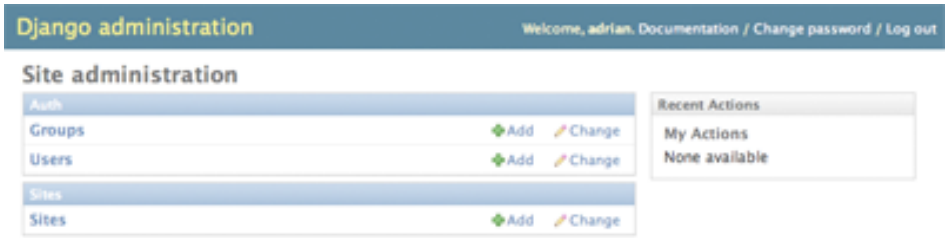
- Add `django.contrib.admin` to `INSTALLED_APPS` in `settings.py`
- Uncomment three lines in `urls.py`:

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()
urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
    # to INSTALLED_APPS to enable admin documentation:
    # (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
)
```

Activating the the admin site

Feast your eyes with the django site!





Registering models and customizing admin forms

- Create a file called `admin.py` in your app directory:

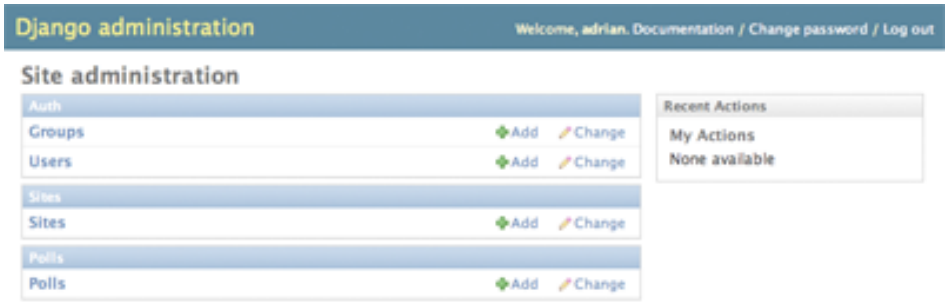
```
from mysite.polls.models import Poll
from django.contrib import admin

admin.site.register(Poll)
```

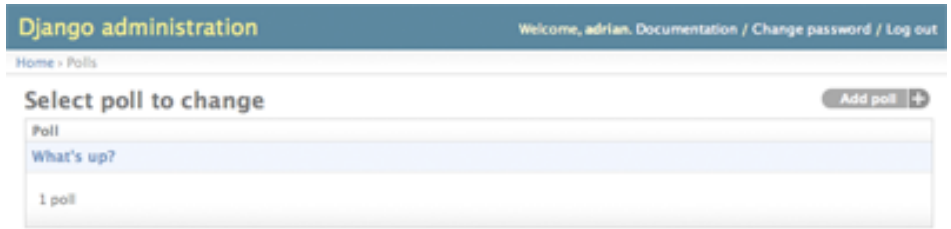
- Customize the admin form:

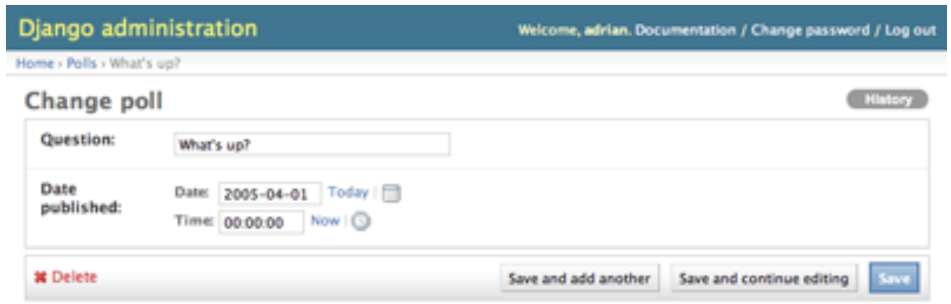
```
class PollAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question']

admin.site.register(Poll, PollAdmin)
```

Registering models and customizing admin forms

Browse/edit your models!







Customization options

1. ModelAdmin media
2. Custom templates
3. ModelAdmin/ModelForm hacking
4. Custom views

ModelAdmin Media

Adding custom media

Add media in the same way as in the `Form` classes:

```
class ArticleAdmin(admin.ModelAdmin):
    class Media:
        css = {
            "all": ("my_styles.css",)
        }
        js = ("my_code.js",)
```

Keep in mind that these will be prepended with `MEDIA_URL`

ModelAdmin Media Examples

- JavaScript
  - WYSIWYG Editor (e.g. TinyMCE)
  - AJAX
  - Fancy Inlines (drag and drop, dynamic add/delete)
    - http://tinyurl.com/add-remove-inlines
    - http://www.djangosnippets.org/snippets/1053/
  - Inject HTML
- CSS
  - Colors
  - Layout

ModelAdmin Media Pros & Cons

- Pros
  - Easy for one-off projects
- Cons
  - Requires JavaScript
  - Only works for the ChangeForm
  - Difficult to bundle as reusable app

Custom Templates

Custom Templates

- `django.contrib.admin` is a “reusable application”
- Key templates
  - `admin/base.html`
  - `admin/index.html`
  - `admin/change_form.html`
  - `admin/change_list.html`

Per Project/App/Model Templates

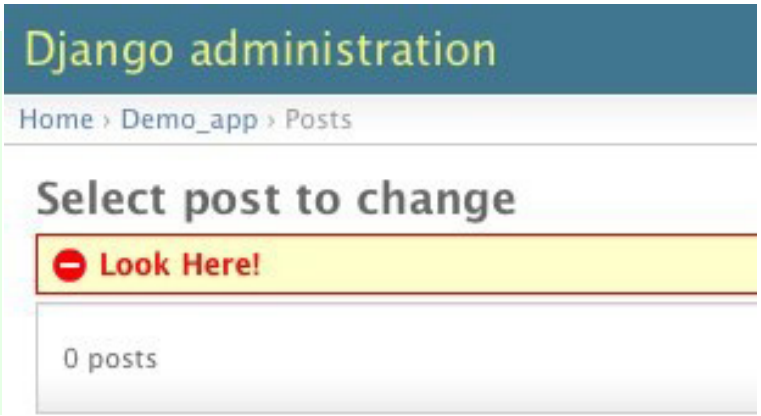
Templates can be overriden:

- Across an entire project:
  `admin/change_form.html`
- Across an application:
  `admin/<my_app>/change_form.html`
- For an individual model:
  `admin/<my_app>/<my_model>/change_form.html`

You can also override some templates by setting relevant attributes on `ModelAdmin`: `change_form_template`, `change_list_template`, `delete_confirmation_template`, `delete_selected_confirmation_template`, `object_history_template`

Custom Template Example

```
{% extends "admin/change_list.html" %}
{% block object-tools %}
 <h1 class="errornote">
  Look Here!
 </h1>
 {{ block.super }}
{% endblock %}
```

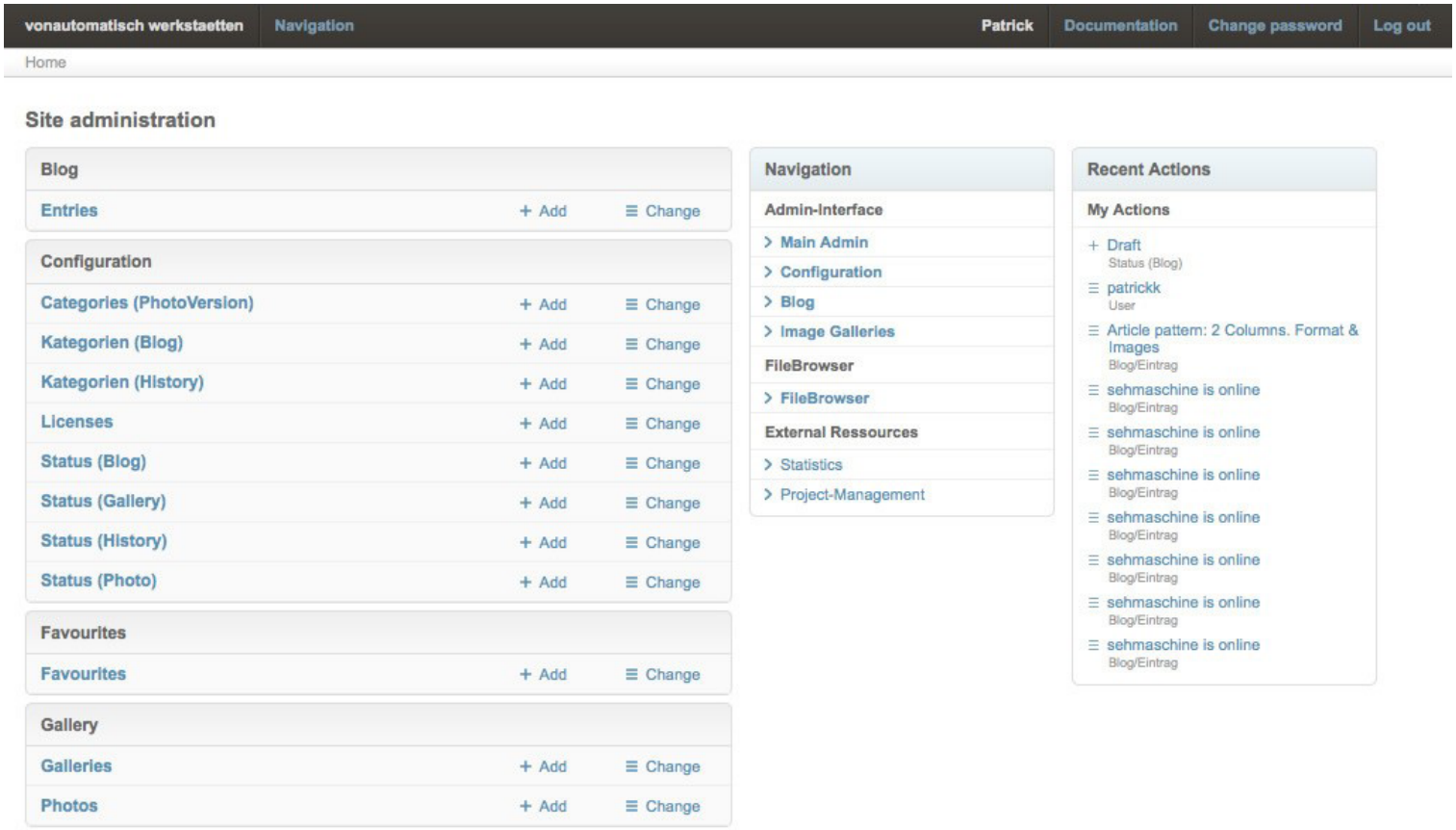


Custom Template Tips

- Extend, don't override
- Use `{{ block.super }}` to extend blocks
- If you need extra blocks, create a customized template and then extend it.
- Extend the `{% extrahead block %}` in `base.html` for admin-wide media

Examples of Custom Templates

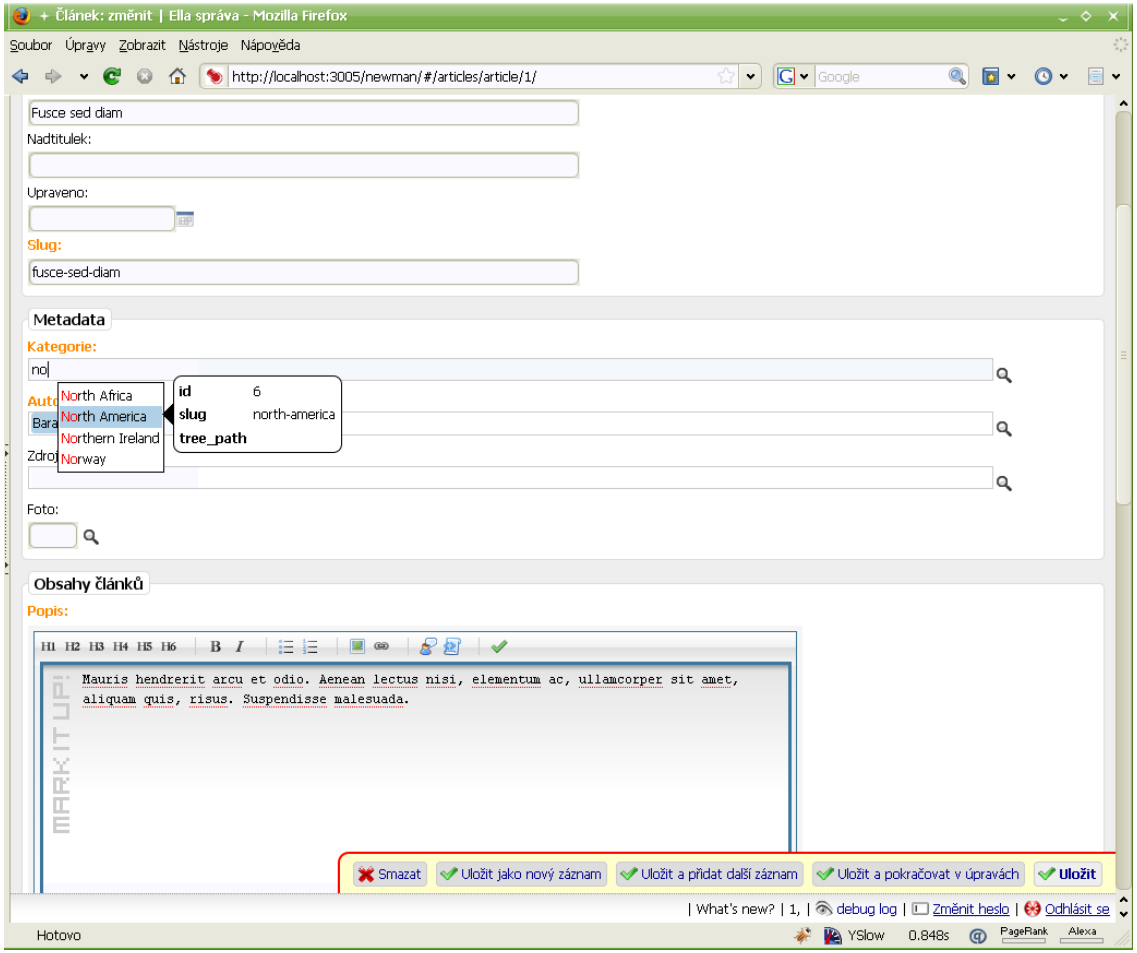
django-grapelli



http://code.google.com/p/django-grappelli/

Examples of Custom Templates

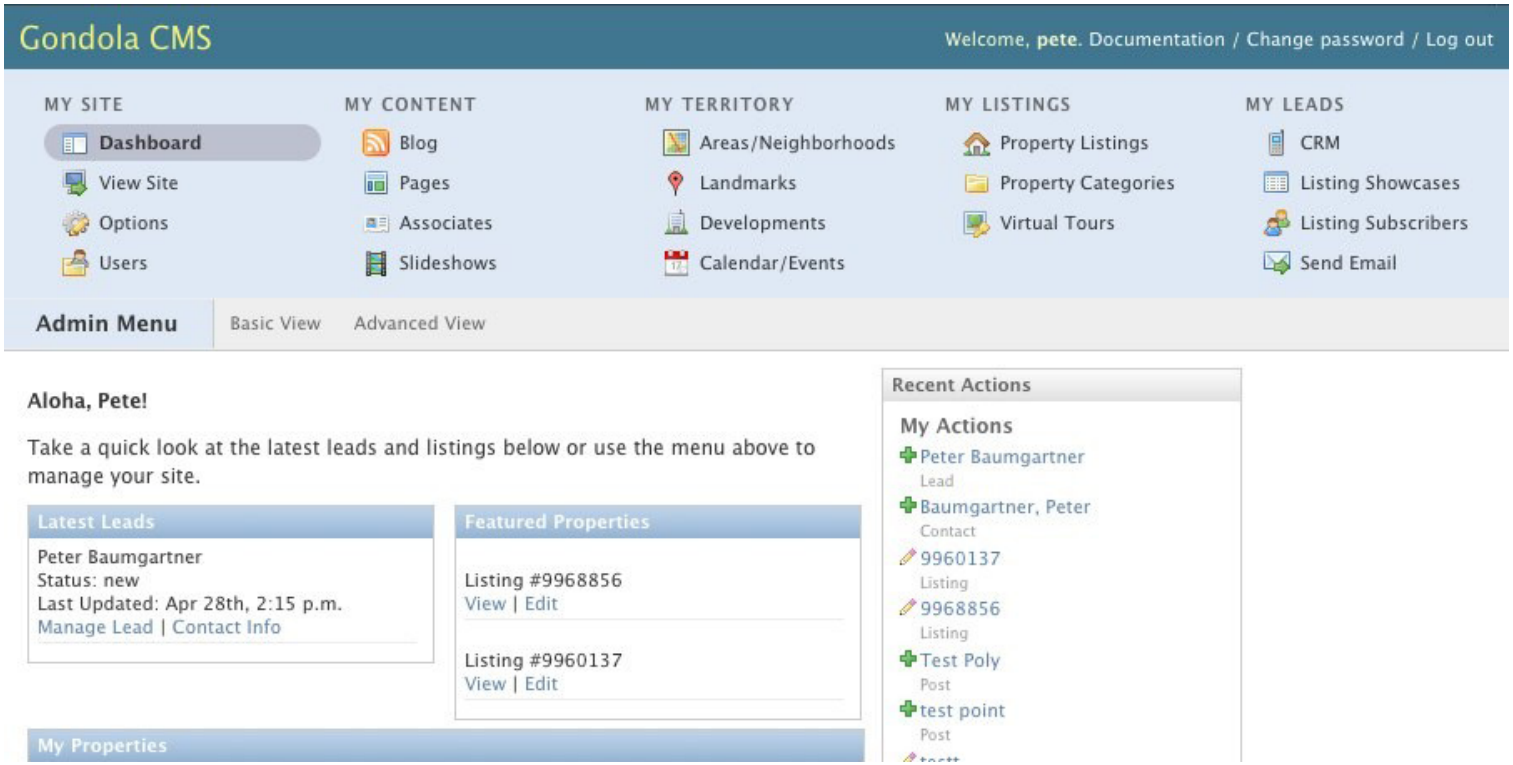
ella



http://github.com/ella/ella

Examples of Custom Templates

gondola



http://gondolacms.com/

Custom Templates Pros & Cons

- Pros
  - Easy
  - Touches every admin view
  - No additional work to bundle with reusable apps
- Cons
  - Mostly cosmetic (not functional) changes

ModelAdmin/ModelForm hacking

ModelAdmin options

- Lots of options in ModelAdmin to override
- Many customizations possible without dwelling into the source code
- Change default fields/fieldsets in change views (`fields`, `fieldsets`)
- Change default search fields/filters in list views (`list_filter`, `search_fields`)
- Change default widgets (`filter_horizontal`, `filter_vertical`)
- Change default ordering (`ordering`)
- Edit related models (`inlines`)
- . . . and many more!

Re-registering a ModelAdmin

It is easy to extend ModelAdmin subclasses from other apps, e.g. `django.contrib.auth`:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from demo_app.models import UserProfile

class UserProfileInline(admin.TabularInline):
    model = UserProfile
    fk_name = 'user'
    max_num = 1

class CustomUserAdmin(UserAdmin):
    inlines = [UserProfileInline, ]

admin.site.unregister(User)
admin.site.register(User, CustomUserAdmin)
```

ModelAdmin methods

- There is a number of methods in ModelAdmin that are used to process requests
- Extend these methods to add functionalities
- Remember that you can add attributes to `request` at runtime
- Methods used to render admin pages:
  - `ModelAdmin.add_view(self, request, form_url='', extra_context=None)`
  - `ModelAdmin.change_view(self, request, object_id, extra_context=None)`
  - `ModelAdmin.changelist_view(self, request, extra_context=None)`
  - `ModelAdmin.delete_view(self, request, object_id, extra_context=None)`
  - `ModelAdmin.history_view(self, request, object_id, extra_context=None)`

Augment the context data

```
class MyModelAdmin(admin.ModelAdmin):

    # A template for a very customized change view:
    change_form_template = 'admin/myapp/extras/openstreetmap_change_form.html'

    def get_osm_info(self):
        # ...

    def change_view(self, request, object_id, extra_context=None):
        my_context = {
            'osm_data': self.get_osm_info(),
        }
        return super(MyModelAdmin, self).change_view(request, object_id,
            extra_context=my_context)
```

Row-level Permissions

```
class ArticleAdmin(admin.ModelAdmin):

    def save_model(self, request, obj, form, change):
        obj.user = request.user
        obj.save()

    def queryset(self, request):
        qs = self.model._default_manager.filter(user=request.user)
        return qs
```

ModelForms

- Much of ModelAdmin's functionality is a wrapper around ModelForm
- If you can't do it in ModelAdmin, chances are ModelForm can help
- Pulled directly from `django.forms` and no different in functionality

ModelForms Example

```
class AuthorForm(forms.ModelForm):
    exclude_states = ['AS', 'GU', 'MP', 'VI',]

    def __init__(self, *args, **kwargs):
        super(AuthorForm, self).__init__(*args, **kwargs)
        w = self.fields['state'].widget
        choices = []
        for key, value in w.choices:
            if key not in self.exclude_states:
                choices.append((key, value))
        w.choices = choices

class AuthorAdmin(admin.ModelAdmin):
    form = AuthorForm
```

ModelAdmin/ModelForm Tips

- The further you dig, the less documentation you'll find
- Don't be afraid to study the source:
  - `django.contrib.admin.sites.AdminSite`
  - `django.contrib.admin.options.ModelAdmin`
  - `django.forms.models.ModelForm`
  - `django.contrib.admin.options.InlineModelAdmin`
  - `django.forms.formsets`
- Use a debugger for sanity (`ipdb.set_trace()`)

ModelAdmin/ModelForm Pros & Cons

- Pros
  - Flexible
  - Powerful
  - No additional work to bundle with reusable apps
- Cons
  - Gets complex quickly
  - May require getting familiar with undocumented Django internals

Custom views

Adding custom admin views

- The admin just wasn't built to do some things
- Other things simply aren't worth the trouble
- Build your own view and plug it into the admin!

Just override the `get_urls()` method on your ModelAdmin subclass:

```
class MyModelAdmin(admin.ModelAdmin):
    def get_urls(self):
        urls = super(MyModelAdmin, self).get_urls()
        my_urls = patterns('',
            (r'^my_view/$',
             self.admin_site.admin_view(self.my_view))
        )
        return my_urls + urls
```

Notice that the custom view is wrapped to protect it from unauthorized access.

Adding Custom Admin Views

Don't be afraid to base your code on default admin views!

```
    def my_view(self, request, model_admin):
        opts = model_admin.model._meta
        admin_site = model_admin.admin_site
        has_perm = request.user.has_perm(opts.app_label \
                    + '.' + opts.get_change_permission())
        context = {'admin_site': admin_site.name,
            'title': "My Custom View",
            'opts': opts,
            'root_path': '/%s' % admin_site.root_path,
            'app_label': opts.app_label,
            'has_change_permission': has_perm}
        template = 'admin/demo_app/my_view.html'
        return render_to_response(template, context,
                context_instance=RequestContext(request))
```

Custom View Template

```
{% extends "admin/base_site.html" %}
{% load i18n %}
{% block breadcrumbs %}
<div class="breadcrumbs">
  <a href="../../../">{% trans "Home" %}</a> &rsaquo;
  <a href="../../">{{ app_label|capfirst|escape }}</a> &rsaquo;
  {% if has_change_permission %}<a
      href="../">{{ opts.verbose_name_plural|capfirst }}</a>
  {% else %}
      {{ opts.verbose_name_plural|capfirst }}
  {% endif %} &rsaquo; My Custom View
</div>
{% endblock %}
{% block content %}
<!-- do stuff here -->
{% endblock %}
```

Custom View Pros & Cons

- Pros
  - More flexible
  - More powerful
  - No additional work to bundle with reusable apps
- Cons
  - Can be tricky to integrate into workflow
  - You're on your own to validate forms, build templates, etc.

Multiple admin sites

Multiple Admin Sites

- `django.contrib.admin.sites.AdminSite`
- An AdminSite object encapsulates an instance of the Django admin application, ready to be hooked in to your URLconf.
- Create your own admin sites for less-experienced users:

```
from django.conf.urls.defaults import *
from myproject.admin import basic_site, advanced_site

urlpatterns = patterns('',
    ('^basic-admin/', include(basic_site.urls)),
    ('^advanced-admin/', include(advanced_site.urls)),
)
```

Reversing Admin URLS

- Name your admin sites to allow URL reversing:

```
from django.contrib.admin.sites import AdminSite
class BasicAdminSite(AdminSite):
    ...

class AdvancedAdminSite(AdminSite):
    ...

basic_site = BasicAdminSite(name='basic')
advanced_site = AdvancedAdminSite(name='advanced')
```

- Use namespaces to reverse URLS:

```
>>> from django.core import urlresolvers
>>> c = Choice.objects.get(...)
>>> change_url=urlresolvers.reverse('admin:polls_choice_change',
... args=(c.id,))
>>> basic_change_url=urlresolvers.reverse('basic:polls_choice_change',
... args=(c.id,))
```

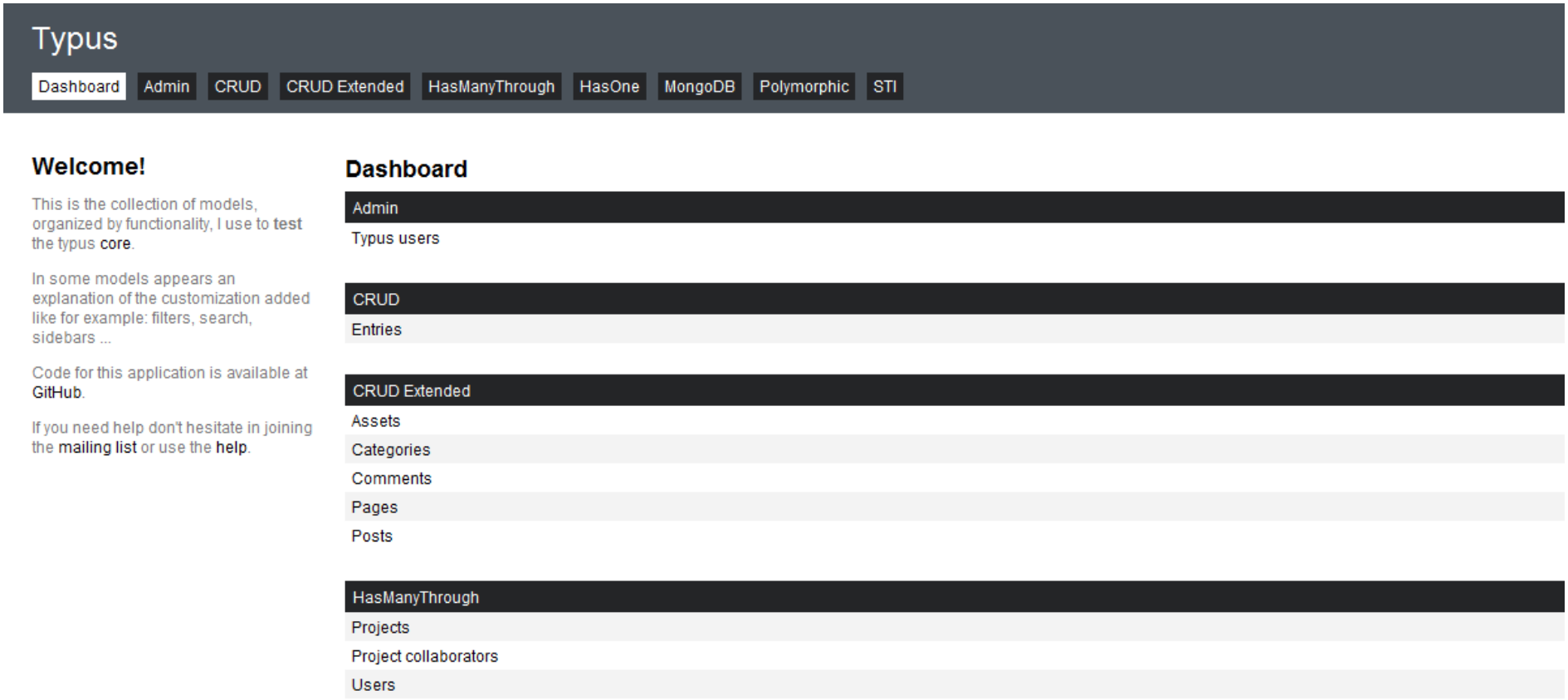
Rails admin

Introduction

- No particular admin module provided with Rails framework
- Scaffolding allows to generate simple admin sites
- A lot of Rails admin gems and plugins

Rails admin examples

Typus



http://docs.typuscmf.com

Rails admin examples

active_scaffold

User Management: users roles permissions

Users Search Create New

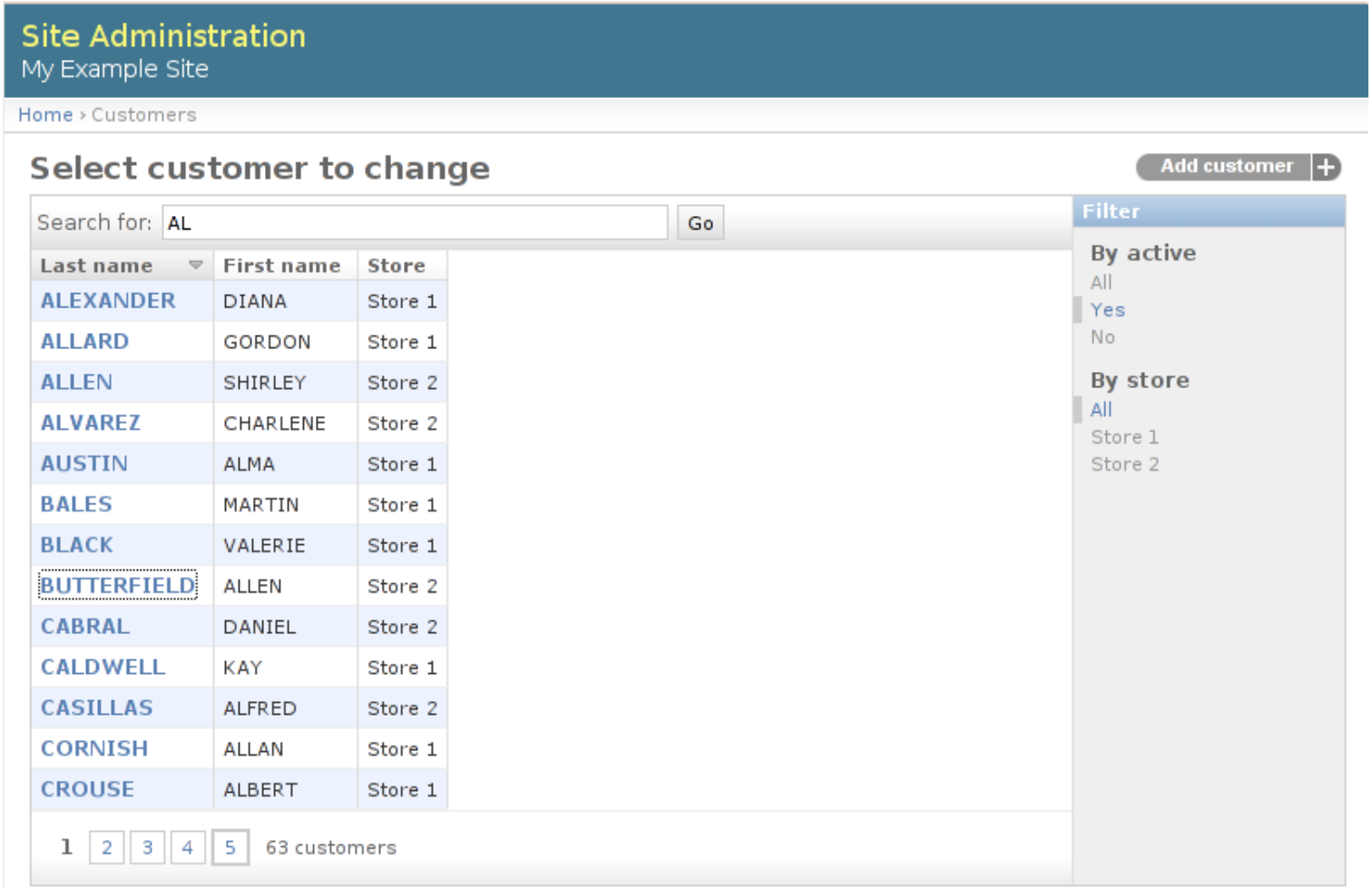
Id	Name	Photo	Email address	Login▲	Roles	Aliases	Addresses	
2143	asdf asdf	-	asdf	asdf	34234, asdf	kijuhy	89 foobar st fooville, 3000, asdf UT, 84321	Names Edit Delete Show
2137	Danny theMan	-	aaa	dan	Testando, Other, adsg, ...	brown, red, yellow	berlin berlin, 10999	Names Edit Delete Show

2 Found

http://activescaffold.com/

Rails admin examples

AutoAdmin



http://code.trebex.net/auto-admin

References

- http://lincolnloop.com/blog/2009/jun/22/customizing-django-admin-eurodjangocon-2009/
- Django documentation: http://www.djangoproject.com/
- Djangobook: http://www.djangobook.com/