Django

"The Web framework for perfectionists with deadlines"

Jacob Poulsgaard Tjørnholm

Agenda

- Introduction
- Models
- Routing, views and templates
- The admin interface
- Miscellaneous

What is Django?

- MVC-style Web-development framework
  - Actually, Model-Template-View (MTV)
- Written in Python
- Biased toward performance
- Well-documented

Background

- Started at World Company (Lawrence, Kansas)
- Used on large news sites
  - lawrence.com
  - washingtonpost.com
- Now BSD-license
- Named after guitarist Django Reinhardt
- Started by Adrian Holovaty et al.

Terminology

The main purpose of a project is to group apps.

- “A project is an instance of a certain set of Django apps, plus the configuration for those apps.”
- “An app is a portable set of Django functionality, usually including models and views, that lives together in a single Python package.”

The “project”

- Defines settings
  - DB connection params
  - Template dir
  - Root URL conf
  - **Apps**

The “app”

- A reusable block of functionality
- Contains models, views, template tags...
- Examples:
  - django.contrib.admin
  - django.contrib.auth
  - reviewboard.iphone
  - reviewboard.diffviewer

```
books/
    __init__.py
    models.py
    views.py
```

Example project

- A simple book database
- Contains a single app: “books”
- Model:
  - Book
  - Author
  - Publisher

Agenda

- Introduction
- Models
- Routing, views and templates
- The admin interface
- Miscellaneous

Models

- Must live inside an app
- Written in Python, not extracted from DB
- More than just data

Models

books/
models.py

```python
from django.db import models

class Publisher(models.Model):
    name = models.CharField(maxlength=30)
    address = models.CharField(maxlength=50)
    city = models.CharField(maxlength=60)
    state_province = models.CharField(maxlength=30)
    country = models.CharField(maxlength=50)
    website = models.URLField()

class Author(models.Model):
    salutation = models.CharField(maxlength=10)
    first_name = models.CharField(maxlength=30)
    last_name = models.CharField(maxlength=40)
    email = models.EmailField()
    headshot = models.ImageField(upload_to='/tmp')

class Book(models.Model):
    title = models.CharField(maxlength=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
```

Models - DB syncing

- manage.py validate
  - Validate the integrity of all models
- manage.py sqlall *app*
  - Output table-creating SQL for *app* on stdout
- manage.py syncdb
  - Create tables for all apps that don't have them already
  - Always a safe operation

Resulting SQL (1/2)

```
BEGIN;
CREATE TABLE "books_publisher" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(30) NOT NULL,
    "address" varchar(50) NOT NULL,
    "city" varchar(60) NOT NULL,
    "state_province" varchar(30) NOT NULL,
    "country" varchar(50) NOT NULL,
    "website" varchar(200) NOT NULL
)
;
CREATE TABLE "books_book" (
    "id" integer NOT NULL PRIMARY KEY,
    "title" varchar(100) NOT NULL,
    "publisher_id" integer NOT NULL REFERENCES "books_publisher" ("id"),
    "publication_date" date NOT NULL
)
;
```

Resulting SQL (2/2)

```
CREATE TABLE "books_author" (
    "id" integer NOT NULL PRIMARY KEY,
    "salutation" varchar(10) NOT NULL,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(40) NOT NULL,
    "email" varchar(75) NOT NULL,
    "headshot" varchar(100) NOT NULL
)
;
CREATE TABLE "books_book_authors" (
    "id" integer NOT NULL PRIMARY KEY,
    "book_id" integer NOT NULL REFERENCES "books_book" ("id"),
    "author_id" integer NOT NULL REFERENCES "books_author" ("id"),
    UNIQUE ("book_id", "author_id")
)
;
CREATE INDEX "books_book_publisher_id" ON "books_book" ("publisher_id");
COMMIT;
```

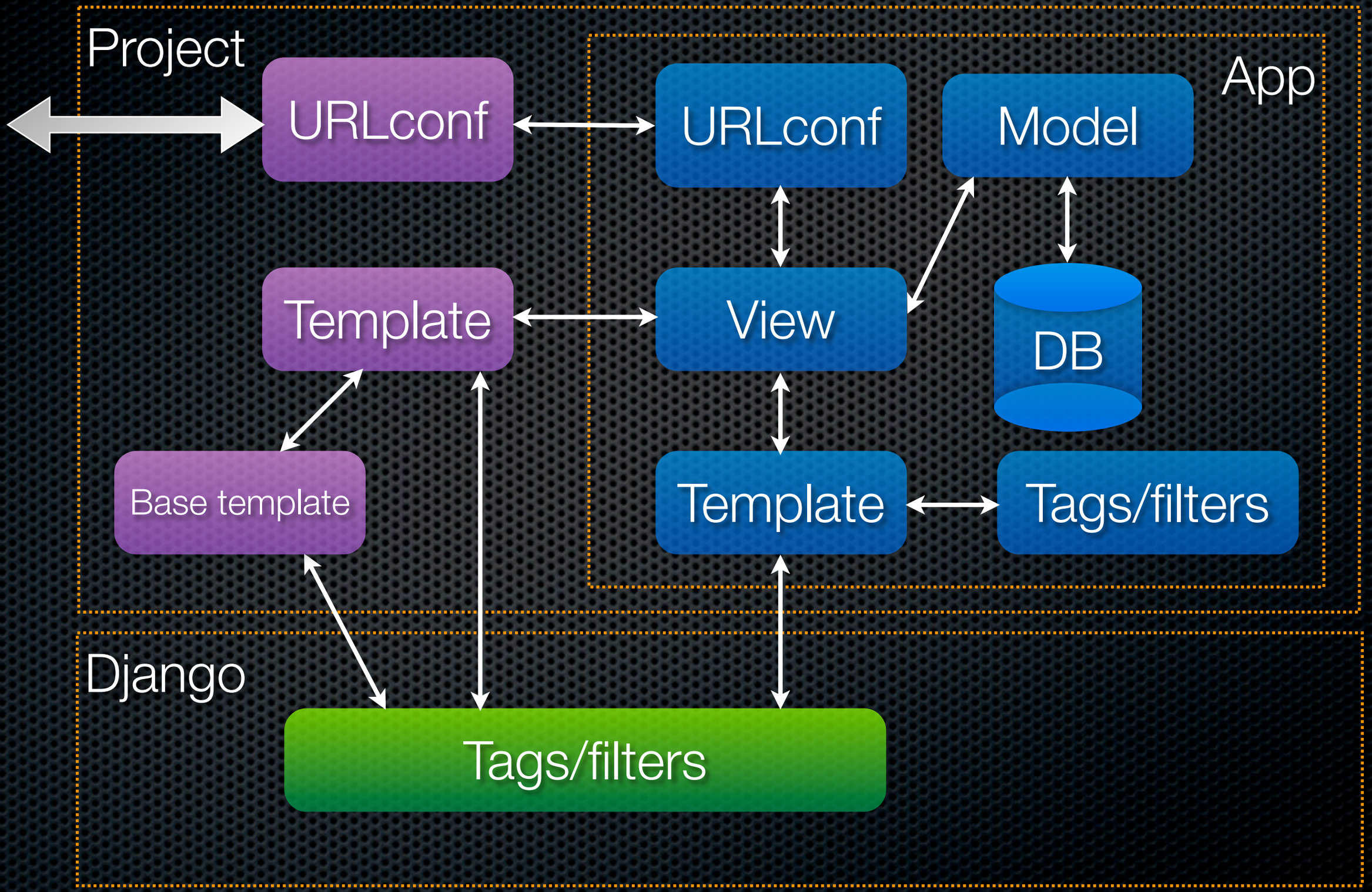
Basic data access

```
>>> from books.models import Publisher
>>> p = Publisher(name='Apress', address='2560 Ninth St.',
...     city='Berkeley', state_province='CA', country='U.S.A.',
...     website='http://www.apress.com/')
>>> p.save()
>>> p = Publisher(name="O'Reilly", address='10 Fawcett St.',
...     city='Cambridge', state_province='MA', country='U.S.A.',
...     website='http://www.oreilly.com/')
>>> p.save()
>>> publisher_list = Publisher.objects.all()
>>> publisher_list
[<Publisher: Publisher object>, <Publisher: Publisher object>]
```

Agenda

- Introduction
- Models
- Routing, views and templates
- The admin interface
- Miscellaneous

Sample request roundtrip



Routing requests to views

"URLconf"

```
urlpatterns = patterns('',
    # Example:
    (r'^now/$', current_datetime),
    (r'^now/plus(1)hour/$', hours_ahead),
    (r'^now/plus([2-9])hours/$', hours_ahead),
    (r'^now/plus(\d{2})hours/$', hours_ahead),

    (r'^now/minus(\d{1,2})hours/$', hours_behind),

    (r'^musicians/$', musicians),

    # Uncomment this for admin:
    (r'^admin/', include('django.contrib.admin.urls')),
)
```

urls.py

Routing URL groups to apps

```python
# urls.py
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    (r'^weblog/', include('mysite.blog.urls')),
    (r'^photos/', include('mysite.photos.urls')),
    (r'^about/$', 'mysite.views.about'),
)

# blog/urls.py
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    (r'^(\d\d\d\d)/$', 'mysite.blog.views.year_detail'),
    (r'^(\d\d\d\d)/(\d\d)/$', 'mysite.blog.views.month_detail'),
)

# "/weblog/2007/" -> mysite.blog.views.year_detail
```

Views

- Simple Python methods
- Determine what is displayed

Simple view functions

```
from django.template.loader import get_template
from django.template import Context
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    t = get_template('current_datetime.html')
    html = t.render(Context({'current_date': now}))
    return HttpResponse(html)

def hours_ahead(request, offset):
    offset = int(offset)
    dt = datetime.datetime.now() + datetime.timedelta(hours=offset)
    html = "<html><body>In %s hour(s), it will be %s.</body></html>" % (offset, dt)
    return HttpResponse(html)
```

Templates

- Decide how the available data is presented
- Very limited by design
- Simple looping and branching
- (Custom) tags and filters
- No python code! (!= erb)

Simple template

```
<html><body>It is now {{ current_date }}.</body></html>
```

Template features

- Variables:
  - {{ person.name }} is {{ person.age }} years old.
  - Item 2 in the list is {{ items.2 }}
- Looping tags:
  - {% for item in item_list %} ... {% endfor %}
- Filters:
  - {{ ship_date|date:"F j, Y" }}
  - {{ name|lower }}
  - {{ my_text|escape|linebreaks }}

More interesting template

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <link rel="stylesheet" href="default.css" type="text/css">
    <title>My to-do list</title>
</head>
<body>
    <h1 id="top">Latest tasks</h1>
    {% if task_list %}
        <ul>
        {% for task in task_list %}<li>{{ task }}</li>{% endfor %}
        </ul>
    {% else %}
        <p>You have no tasks.</p>
    {% endif %}
    <hr>
    <p><a href="#top">Back to top</a>.</p>
</body>
</html>
```

Template inheritance

- “Inside-out server side includes”
- Base templates are skeletons for child templates to fill in
- Children define only the blocks they want - the parent always provides a default

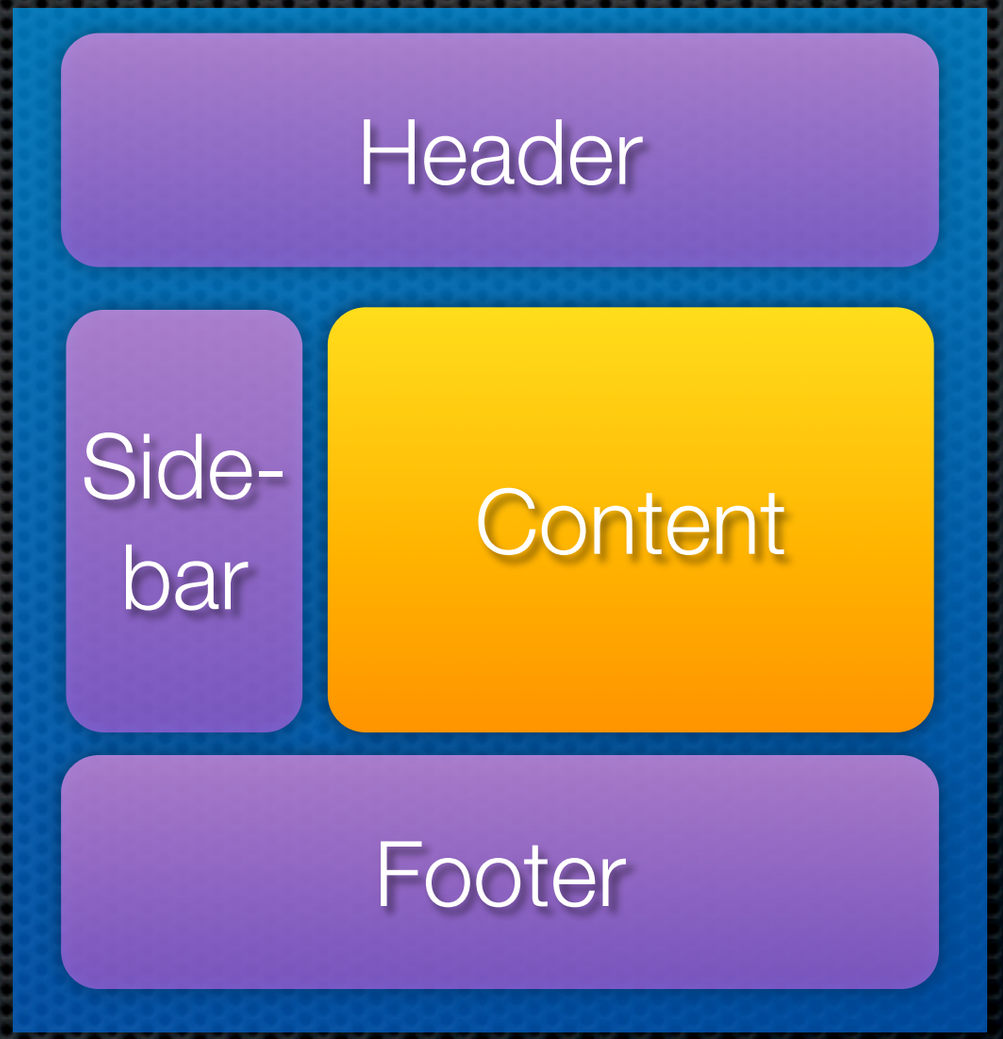
Template inheritance - motivation

Consider traditional server-side include methods

header.php

sidebar.php

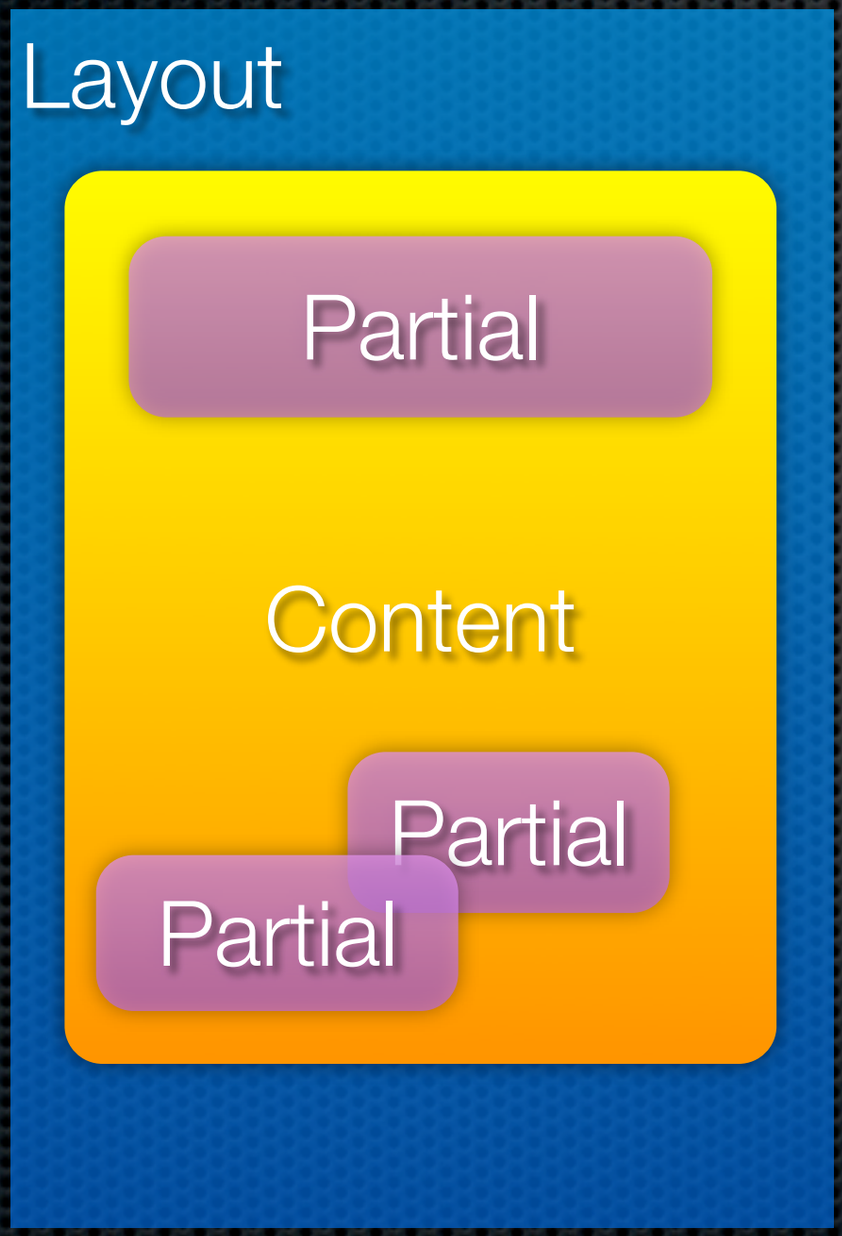
footer.php



my_page.php

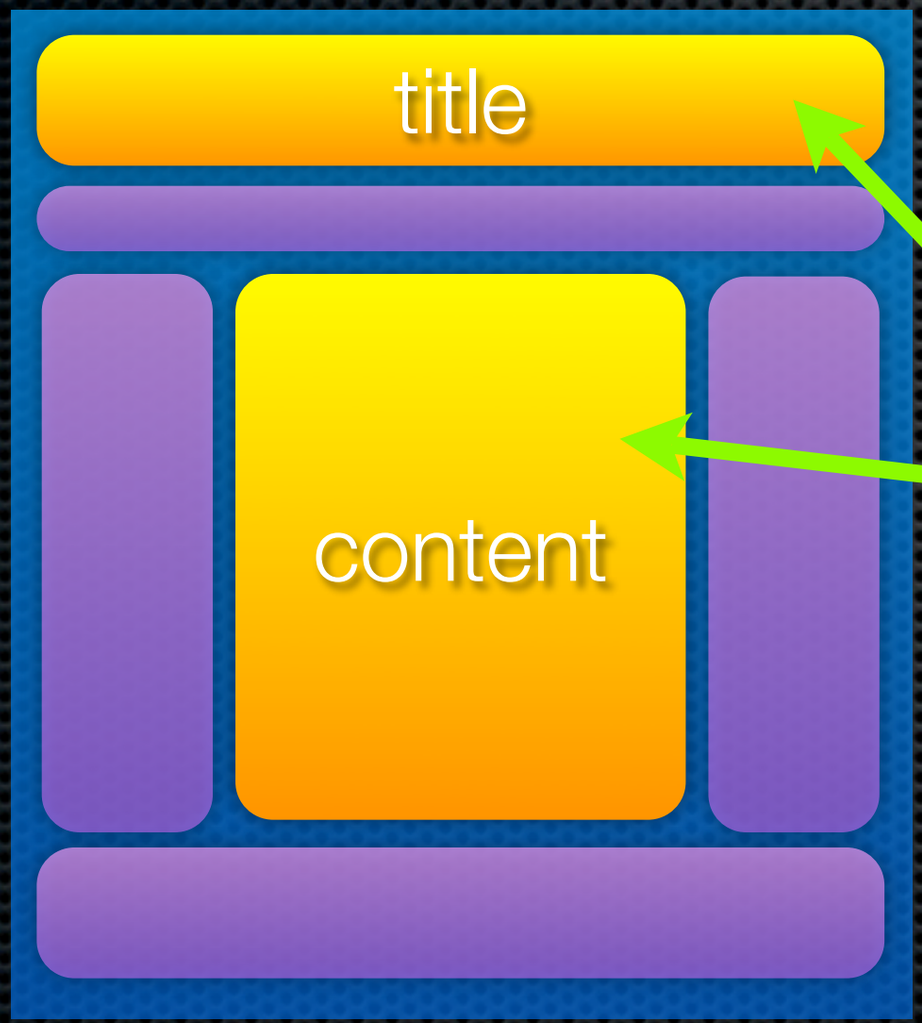
Template inheritance - motivation

...or Rails techniques



Template inheritance - motivation

The Django approach:



base.html

```
{% extends "base.html" %}

{% block title %}The current time{% endblock %}

{% block content %}
<p>It is now {{ current_date }}.</p>
{% endblock %}
```

current_time.html

Base template (base.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    {% block content %}{% endblock %}
    {% block footer %}
    <hr>
    <p>Thanks for visiting my site.</p>
    {% endblock %}
</body>
</html>
```

Child templates

```
{% extends "base.html" %}

{% block title %}The current time{% endblock %}

{% block content %}
<p>It is now {{ current_date }}.</p>
{% endblock %}
```

current_datetime.html

```
{% extends "base.html" %}

{% block title %}Future time{% endblock %}

{% block content %}
<p>In {{ offset }} hour(s), it will be {{ next_time }}.</p>
{% endblock %}
```

hours_ahead.html

Generic views

- Pre-written view functions
- A bit like Rails scaffolding
- Templates still written by you
- Just functions - easily wrapped

Generic views - simple

Rendering a template

```
urlpatterns = patterns('django.views.generic.simple',
    (r'^foo/$',               'direct_to_template', {'template': 'foo_index.html'}),
    (r'^foo/(?P<id>\d+)/$', 'direct_to_template', {'template': 'foo_detail.html'}),
)
```

Doing a redirect

```
urlpatterns = patterns('django.views.generic.simple',
    ('^foo/(?p<id>\d+)/$', 'redirect_to', {'url': '/bar/%(id)s/'}),
)
```

Generic views - complex

- More powerful, higher-level abstractions
  - List/detail views
  - Date-based views (“http://x.y.z/year/month/day”)
  - CRUD views
- Uses optional arguments to control behaviour

Agenda

- Introduction
- Models
- Routing, views and templates
- The admin interface
- Miscellaneous

The admin app - motivation

- Writing admin interfaces is
- Boring
- Always the same
- -> Neglected

The admin app

Activation (1/4):

Enable in model

```
class Book(models.Model):
    title = models.CharField(maxlength=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()

class Admin:
    pass
```

The admin app

Activation (2/4):

Add the app to your project

```
INSTALLED_APPS = (
    'test1.books',
    'django.contrib.admin',
)
```

settings.py

The admin app

Activation (3/4):

Create the corresponding tables

```
[ nova - ~/projects/test/django/test1 ] jtj$ ./manage.py syncdb
Creating table django_admin_log
Installing index for admin.LogEntry model
Loading 'initial_data' fixtures...
No fixtures found.
```

The admin app

Activation (3b/4):

What happened?

```
[ nova - ~/projects/test/django/test2 ] jtj$ ./manage.py dbshell
SQLite version 3.4.0
Enter ".help" for instructions
sqlite>
sqlite> .tables
auth_group                      auth_user_user_permissions
auth_group_permissions          django_admin_log
auth_message                    django_content_type
auth_permission                 django_session
auth_user                       django_site
auth_user_groups
sqlite>
```

The admin app

Activation (4/4): URL mapping

urls.py

```
urlpatterns = patterns('',
    (r'^admin/', include('django.contrib.admin.urls')),
)
```

<django>/django/contrib/admin/urls.py

```
urlpatterns = patterns('',
    ('^$', 'django.contrib.admin.views.main.index'),
    ('^r/(\d+)/(.*)/$', 'django.views.defaults.shortcut'),
    ('^jsi18n/$', i18n_view, {'packages': 'django.conf'}),
    ('^logout/$', 'django.contrib.auth.views.logout'),
    ('^password_change/$', 'django.contrib.auth.views.password_change'),
```

The admin app

Data model recap

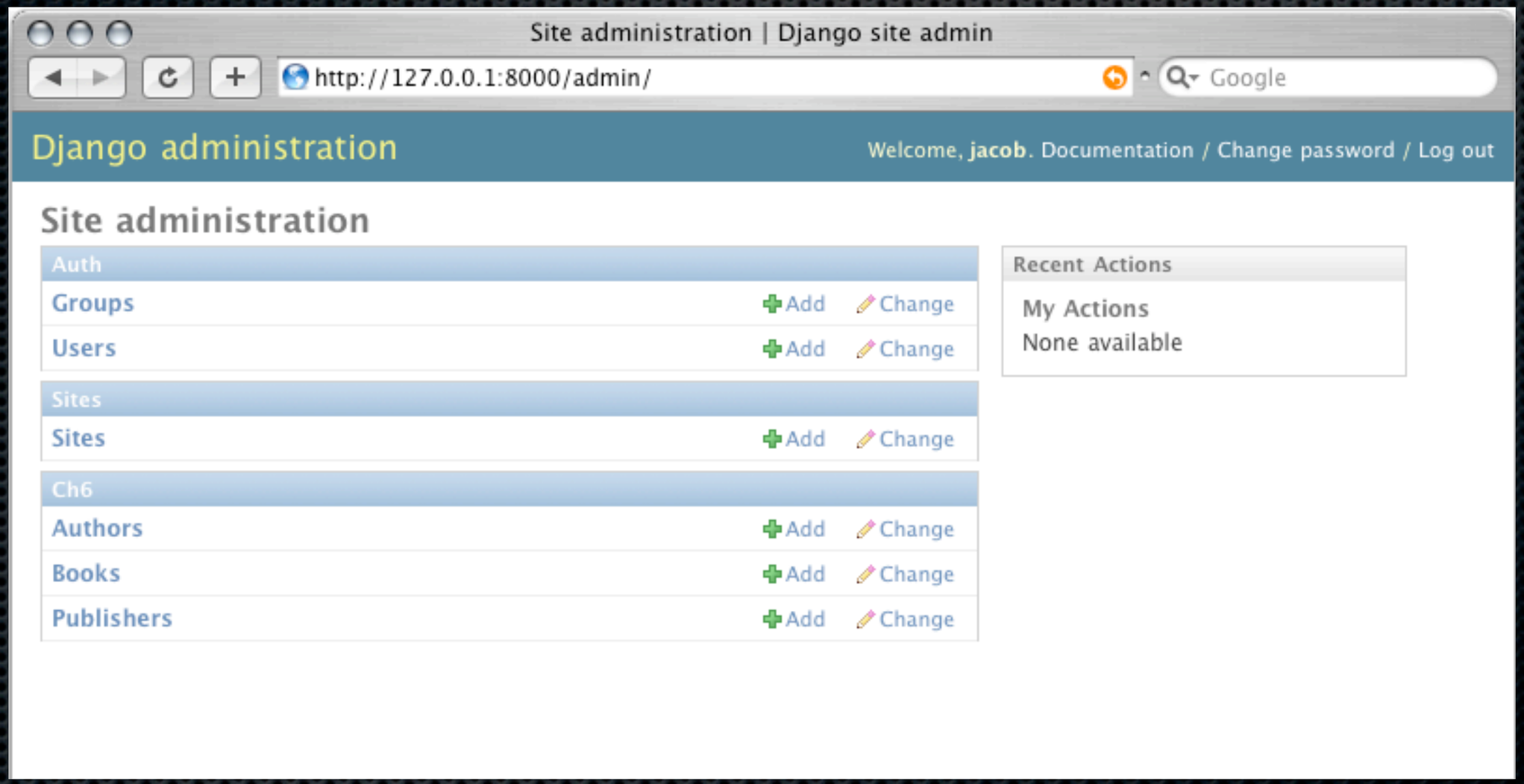
```
class Publisher(models.Model):
    name = models.CharField(maxlength=30)
    address = models.CharField(maxlength=50)
    city = models.CharField(maxlength=60)
    state_province = models.CharField(maxlength=30)
    country = models.CharField(maxlength=50)
    website = models.URLField()

class Author(models.Model):
    salutation = models.CharField(maxlength=10)
    first_name = models.CharField(maxlength=30)
    last_name = models.CharField(maxlength=40)
    email = models.EmailField()
    headshot = models.ImageField(upload_to='/tmp')

class Book(models.Model):
    title = models.CharField(maxlength=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()
```

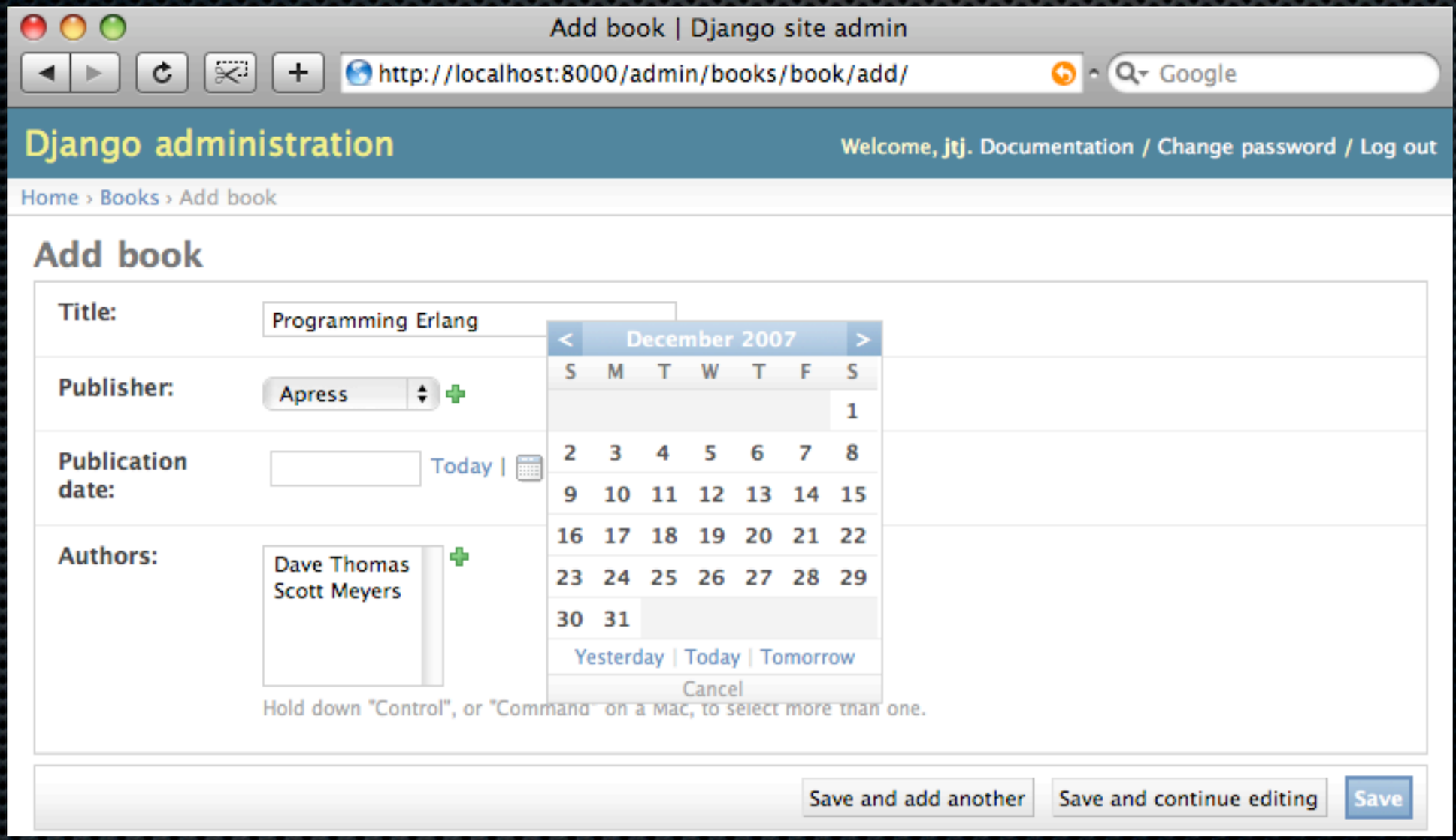
The admin app

Enjoy!



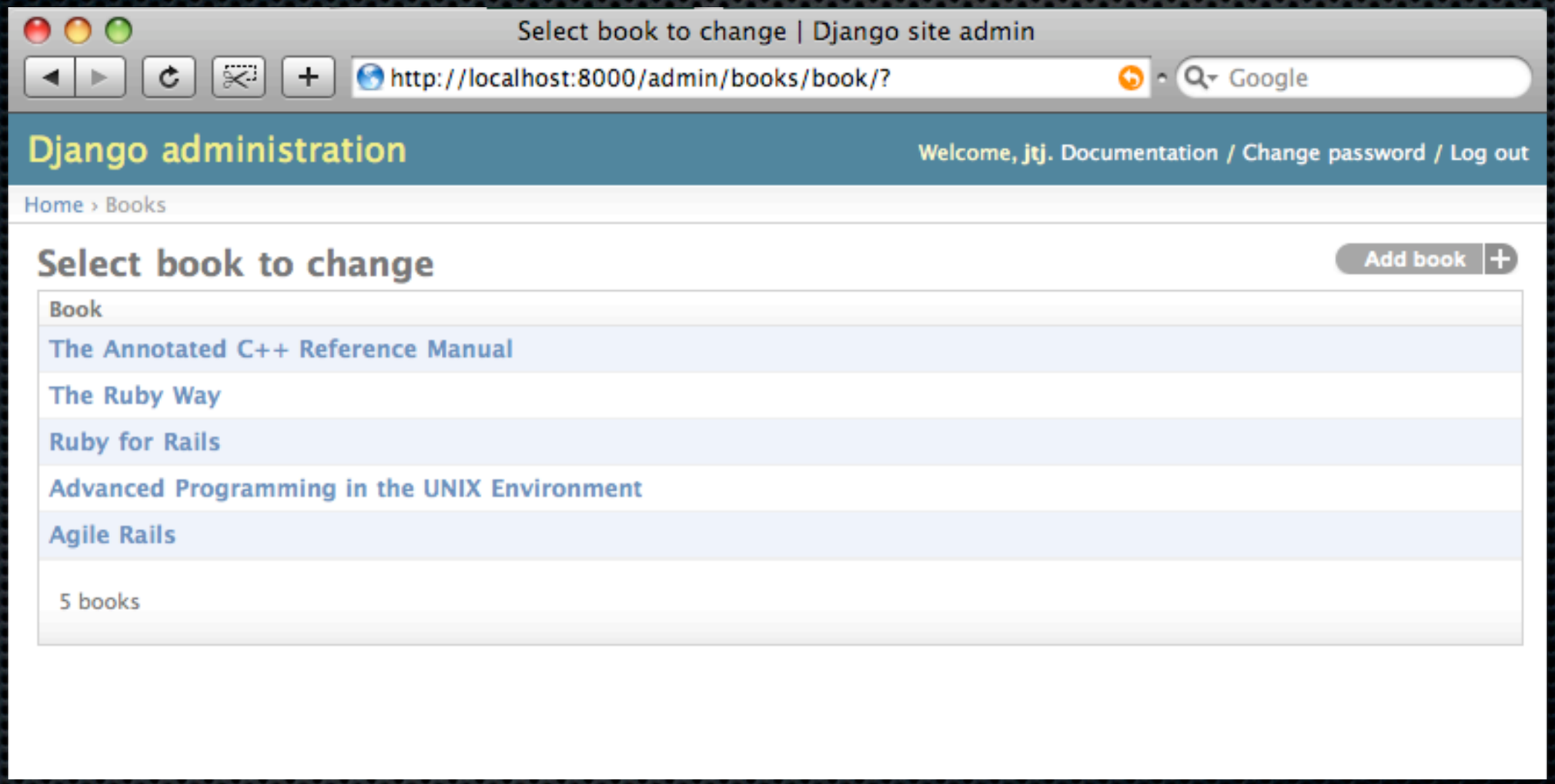
The admin app

Adding/editing books



Customizing admin access

The default admin listing is boring



Customizing admin access

Add properties to the Admin class

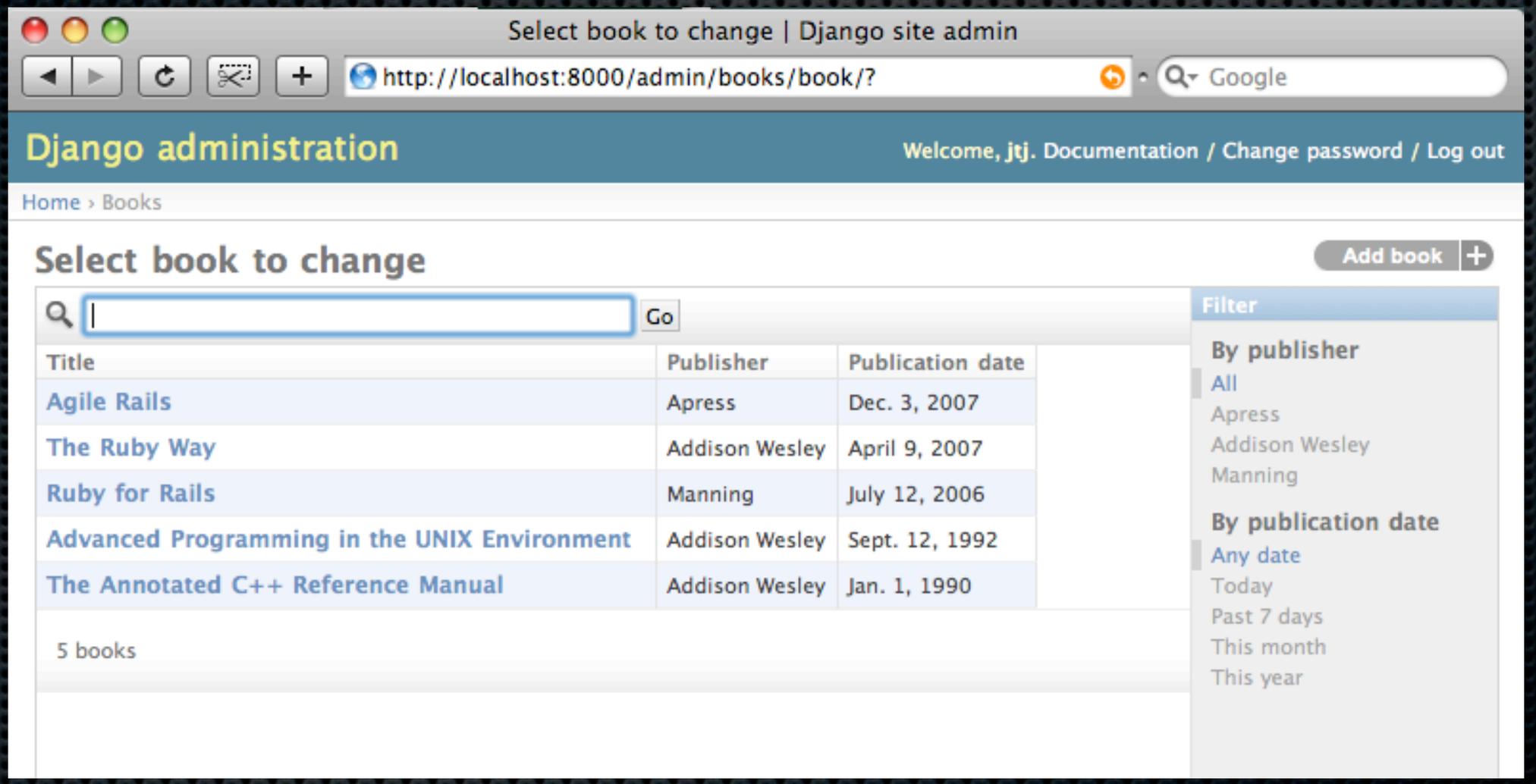
```
class Book(models.Model):
    title = models.CharField(maxlength=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()

    def __str__(self):
        return self.title

    class Admin:
        list_display   = ('title', 'publisher', 'publication_date')
        list_filter    = ('publisher', 'publication_date')
        ordering       = ('-publication_date',)
        search_fields  = ('title',)
```

Customizing admin access

Voila



Agenda

- Introduction
- Models
- Routing, views and templates
- The admin interface
- Miscellaneous

Decorators

Using them

```
@login_required
def new_review_request(request,
                       template_name='reviews/new_review_request.html'):
    if request.method == 'POST':
        form = NewReviewRequestForm(request.POST, request.FILES)
```

```
@register.simple_tag
def pendingreviewcount(obj):
    """
    Returns the pending review count in a list of review requests belonging
    to the specified object.
    """
    return str(obj.reviewrequest_set.filter(public=True, status='P').count())
```

Decorators

Defining them

```
@simple_decorator
def login_required(view_func):
    """Simplified version of auth.decorators.login_required,
       which works with our LOGIN_URL and removes the 'next'
       parameter which we don't need yet.
    """
    def _checklogin(request, *args, **kwargs):
        if request.user.is_authenticated():
            return view_func(request, *args, **kwargs)
        else:
            return HttpResponseRedirect('%s?next_page=%s' % \
                (settings.LOGIN_URL, request.path))
    return _checklogin
```

Tip

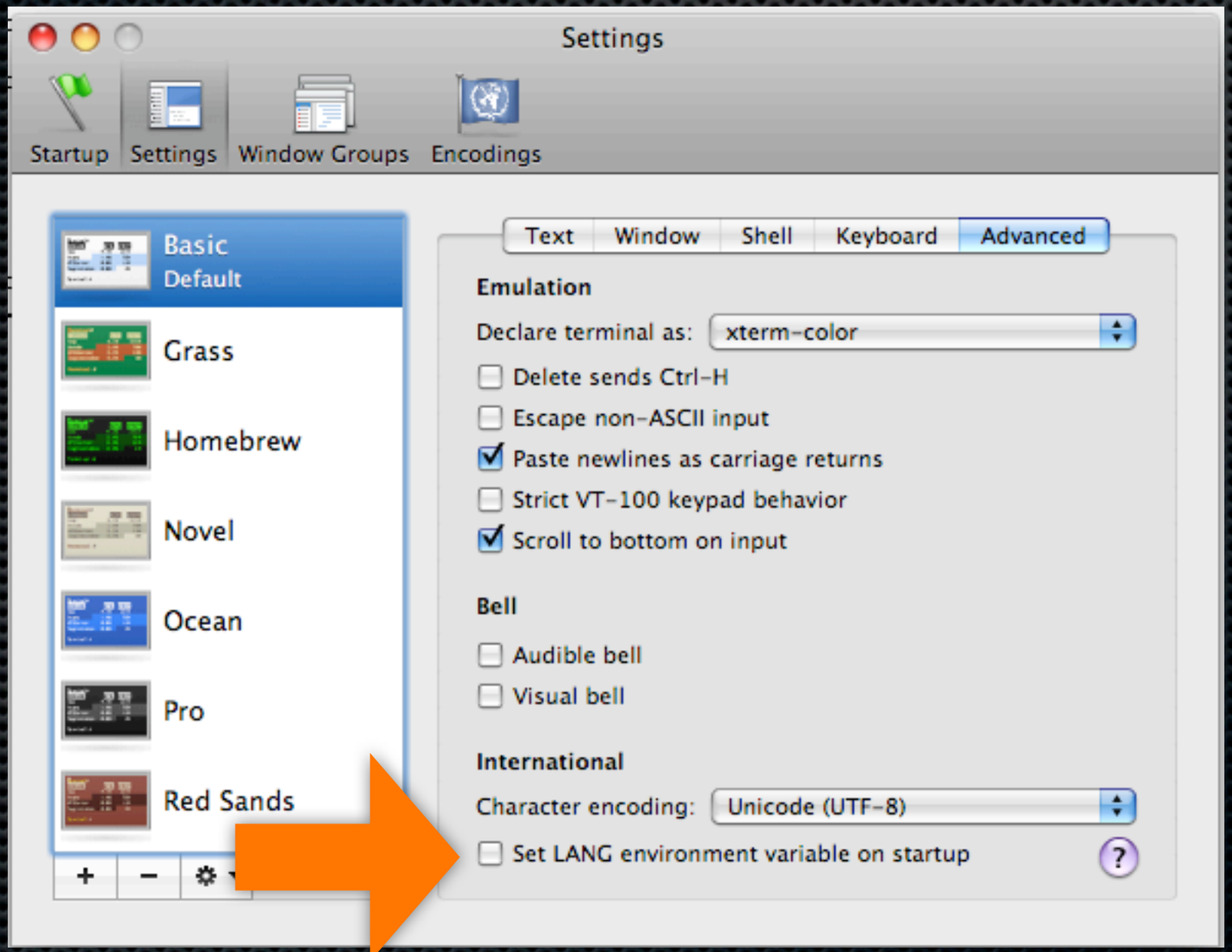
Learn to use manage.py

Located in the root of your project

adminindex
createcachetable
dbshell
diffsettings
dumpdata
flush
inspectdb
loaddata
reset
runfcgi
runserver
shell
sql
sqlall
sqlclear
sqlcustom
sqlflush
sqlindexes
sqlinitialdata
sqlreset
sqlsequencereset
startapp
syncdb
test
testserver
validate

Leopard notes

1. Change Terminal.app settings



Leopard notes

2. Be aware of sys.path issues

- Some installers put files here:
  - /System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/site-packages/
  - /Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/site-packages/
- Neither of these are in the default sys.path
- Workaround: Add them to PYTHONPATH in ~/.bash_profile
  - (Oh, by the way, ~/.bashrc is broken in Leopard)