Django

Python Web Framework

Rayland Jeans

CSCI 5448

"The framework for perfectionists with deadlines"

Overview

- History of Django. How Django got started.
- What is the Django Web Framework?
- Creating a Django Project.
- How Django projects are configured?
- Django's support for MVC.
- The Django Admin Site.
- The benefits of using Django.
- Sites using Django.

Django History

- Django started out as a simple set of tools used by a web team for a newspaper company in Kansas.
- In a couple years, they developed a set of libraries that worked extremely well together.
- The libraries automated or simplified common tasks of web development, that helped them get their work done quickly and efficiently.
- In 2005, they released the libraries under an open source license.
- They named the framework after jazz guitarist Django Reinhardt.

What is the Django Web Framework?

- At Django's core is a set of well-tested python libraries covering all the repetitive tasks experienced during web development, which include:
  - *An object-relational mapper*
  - *Libraries that know how to handle HTTP requests*
  - *A URL routing library that lets you specify the URLs you want to use with your web application.*
  - *A templating system that lets non-programmers write HTML mixed with data.*
  - *A validation library that helps you display forms in web pages*

Creating a Django Project

- Django provides several out-of-the-box commands to create new projects.
- Django projects can be created using the following command:

```
django-admin.py startproject myproject
```

- The previous command generates a python module in the myproject directory with the following files.

```
__init__.py
manage.py
settings.py
urls.py
```

How Django is configured

- Django can be configured using the files generated from the `django-admin.py startproject` command.
- The main configuration file for a Django project is contained in the settings.py module.

How Django is configured

- Basic configurations defined in settings.py
  - INSTALLED_APPS: A tuple of strings indicating all the applications the Django project will run.
  - TEMPLATE_DIRS: A list of directories where the template file will be located.
  - DATABASES: A dictionary containing the settings for all databases to be used in Django.
    - Default supported database backends:
      - PostgreSql, MySql, Sqlite3, Oracle
    - Custom database backends can also be used.

How are Django projects organized?

Sections from sample settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/home/testuser/django-project/cms/cms.db', }
}

TEMPLATE_DIRS = (
    "/home/testuser/django-project/templates"
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'coltrane', # <- Sample application
)
```

Creating a Django Project

- Creating a Django application can be done by running the following command:

```
python manage.py  startapp coltrane
```

- The previous command creates a python module named coltrane with the following files:

```
__init__.py
views.py
models.py
tests.py
```

Django MVC

- Most web applications require the need for the following things:
  - *a way to store and structure data usually known as the model*
  - *a way to present that data, known as the view*
  - *a way to control interaction between the model and the view, know as the controller.*
- These are more commonly known as Model, View Controller (MVC)

Django MVC

- Django provides the same concepts, but prefers to call it Model, Template, View (MTV)
- Django maintains the idea of the Model, but replaces the idea of a Controller with Views.
- Views are then replaced with Templates that can be standard HTML templates with added functionality provided by a Django specific template language.

Django Model

- Data Driven Web development usually requires the need to persist data.
- Django uses an Object Relation Mapper (ORM) to persist the model to a database.
- Django's ORM provides a database-abstraction API that allows developers to create, retrieve, update and delete objects.
- The ORM also removes the need for tedious SQL statements littering the code.

Django Model

- Model classes must inherit from **`django.db.models.Model`** in order to be handled by the built in ORM library.

```
class Entry(models.Model):
    # Core fields
    title = models.CharField(max_length=250)
    excerpt = models.TextField(blank=True) # Optional
    body = models.TextField()

    ... Code Removed ...

    def save(self, force_insert=False, force_update=False):
        self.body_html = markdown(self.body)
        if self.excerpt:
            self.excerpt_html = markdown(self.excerpt)
        super(Entry, self).save(force_insert, force_update)

    def get_absolute_url(self):
        return "/weblog/%s/%s/" % \
            (self.pub_date.strftime("%Y/%b/%d").lower(), self.slug)
```

Django Model

- Classes in the module **`django.db.models`**, such as CharField and TextField can be used to define field types.
- These classes tell Django's ORM how to map the fields to the database.

```
class Entry(models.Model):
    # Core fields
    title = models.CharField(max_length=250)
    excerpt = models.TextField(blank=True) # Optional
    body = models.TextField()

    ... Code removed ...

    def save(self, force_insert=False, force_update=False):
        self.body_html = markdown(self.body)
        if self.excerpt:
            self.excerpt_html = markdown(self.excerpt)
        super(Entry, self).save(force_insert, force_update)
```

Django Model

- Running the following command generates the tables required to persist the model in the database

```
python manage.py syncdb
```

- The previous command will generate the following code to create a table in the database

```
CREATE TABLE "coltrane_entry" (
    "id" integer NOT NULL PRIMARY KEY,
    "title" varchar(250) NOT NULL,
    "excerpt" text NOT NULL,
    "body" text NOT NULL,
);
```

Django Views

- Within a Django application, by convention, views live in the views.py module.
- views.py provide methods called by the framework that will return an HTTP response.

Segment from views.py
This example returns all Entries from the database to be rendered in the response.

```
from django.shortcuts import render_to_response
from coltrane.models import Entry

def entries_index(request):
    return render_to_response(
        'coltrane/entry_index.html',{ 'entry_list' :
        Entry.objects.all() })
```

Django Views

- Views are pretty simple
- They are just functions that return an HTTP response
- There is no naming convention for functions
- Django provides methods to handle common HTTP response functions such as the render_to_response method in the previous example.

Django Views

- To make a view function callable by Django, the function must be mapped to a particular URL.
- URL's are mapped in the urls.py module.
- This mapping provides a way for Django to respond to a URL request and display the response.
- The view will define which template will handle the response.

Django Views

- URL's are mapped using the urls.py module
- Django allows the ability to use regular expressions to define a URL pattern, and map it to a view function.

Segment from urls.py

```
urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^weblog/$', 'coltrane.views.entries_index'),
)
```

Django Templates

- Django templates are designed to be usable by those familiar with HTML.
- Django templates render the content of a response.
- In Django a template is a string that can be combined with data to produce output.
- Django templates contain place holders that are replaced with information from the database and the result is returned as HTML.

Django Templates

- Templates contains variables that are replaced with value when evaluated
- Templates also contains tags which control the logic of the template.
- Templates allows developers to separate logic from presentation.
  - *This approach supports reusable templates*

Django Templates

- Sample template

```
Entries index
{% for entry in entry_list %}
{{ entry.title }}
Published on {{ entry.pub_date|date:"F j, Y" }}
{% if entry.excerpt_html %}
    {{ entry.excerpt_html|safe }}
{% else %}
    {{ entry.body_html|truncatewords_html:"50"|
safe }}
{% endif %}
Read file entry
{% endfor %}
```

Logic Tag

Variable to be replaced with data

End Logic Tag

Django Admin Site

- Django uses a built in Admin interface that is completely customizable.
- Model components can be added to the Admin interface by registering the Class and an Admin Class.
- Model classes are handled using Django's Object Relational Mapper (ORM).
- Provides the ability to use an automatic generated GUI to generate data within the persistence layer.

Django Admin Site

- To use the Django administration application to administer model objects for your application, a python module must be generated that inherits from **`django.contrib.admin.ModelAdmin`**

Sample admin.py

```
from django.contrib import admin
from coltrane.models import Entry

class EntryAdmin(admin.ModelAdmin):
    prepopulated_fields = { 'slug': ['title'] }
```

Django Admin Site

- Django allows developers to customize the admin site by overriding default values.
- The following will auto-populate the a field called **`slug`** with the values from the **`title`** field.

Sample admin.py

```
from django.contrib import admin
from coltrane.models import Entry

class EntryAdmin(admin.ModelAdmin):
    prepopulated_fields = { 'slug': ['title'] }
```

Django Admin Site

- The following code registers the EntryAdmin class with Django's admin site.

Sample admin.py

```
from django.contrib import admin
from coltrane.models import Entry

class EntryAdmin(admin.ModelAdmin):
    prepopulated_fields = { 'slug': ['title'] }

admin.site.register(Entry, EntryAdmin)
```

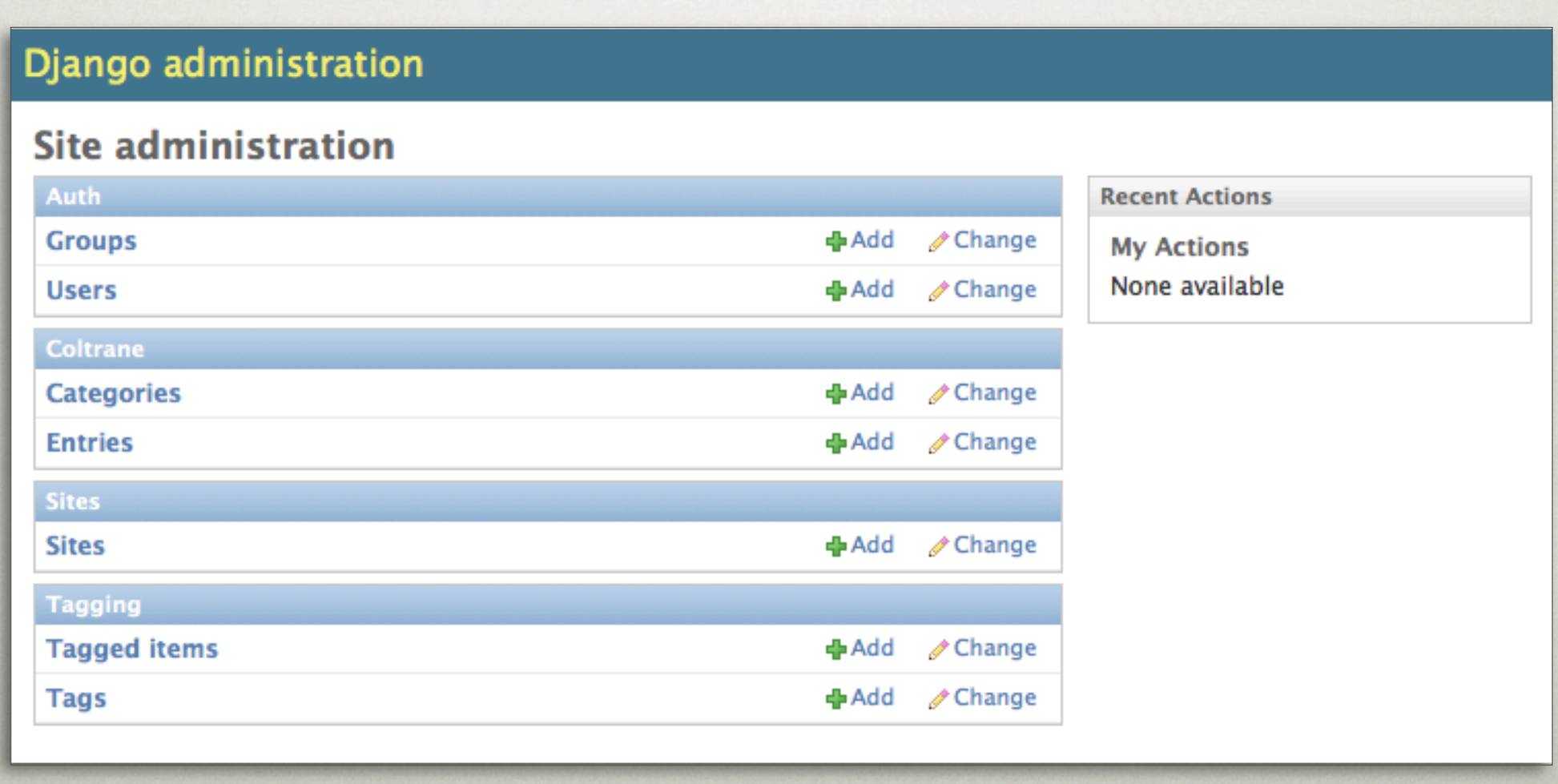
Django Admin Site

The following command will start the Django project

```
python manage.py runserver
```

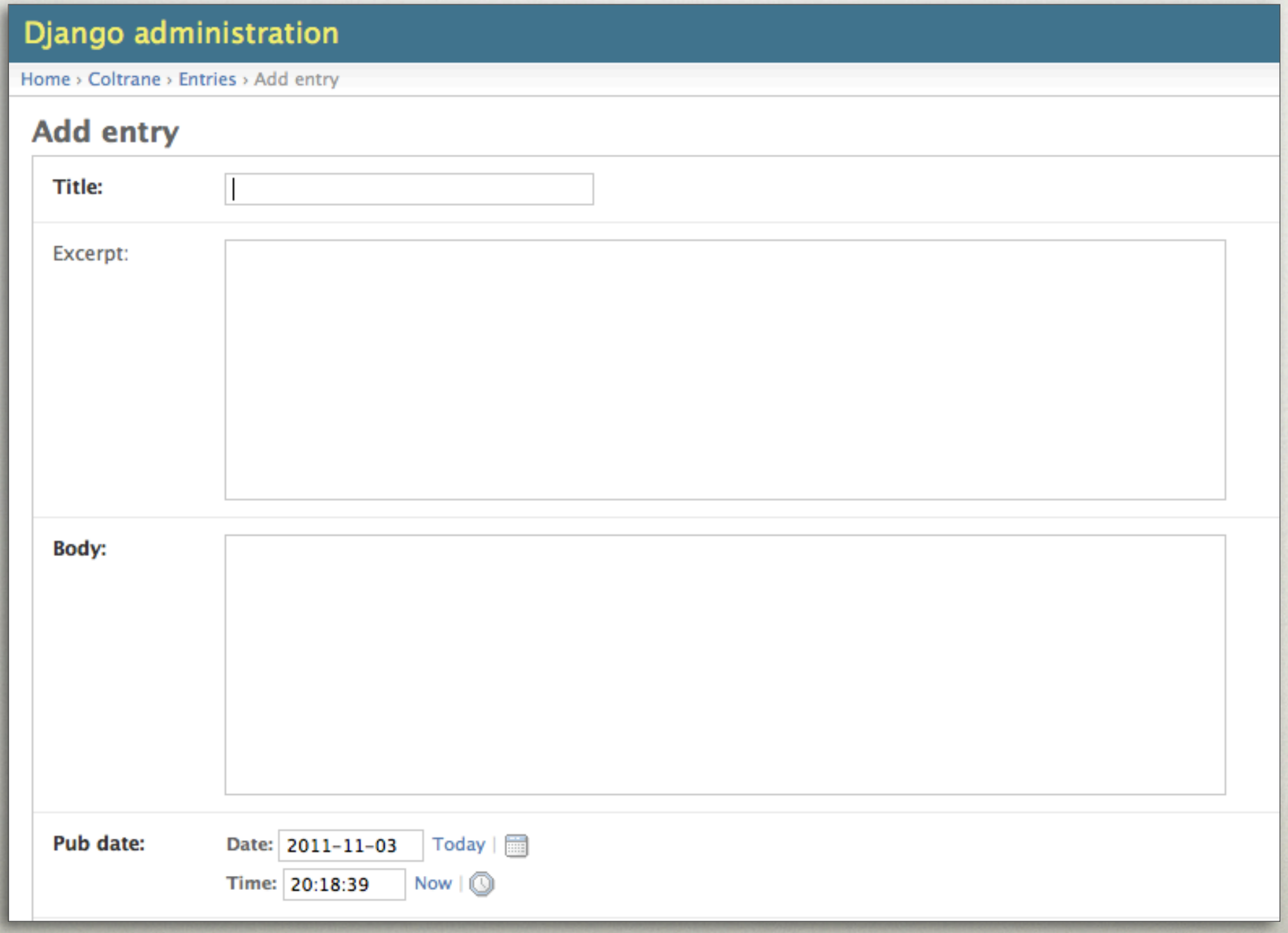
The admin site can be accessed at

http://127.0.0.1:8000/admin



Django Admin Site

Sample Django Admin page to edit Entry objects



Sample Django Application

Sample Application accessed through url mapping:
http://127.0.0.1:8000/weblog

Entries index

Test

Published on October 22, 2011

Some excerpt for the entry

Read file entry

Entry 2

Published on October 22, 2011

This is the excerpt for Entry 2

Read file entry

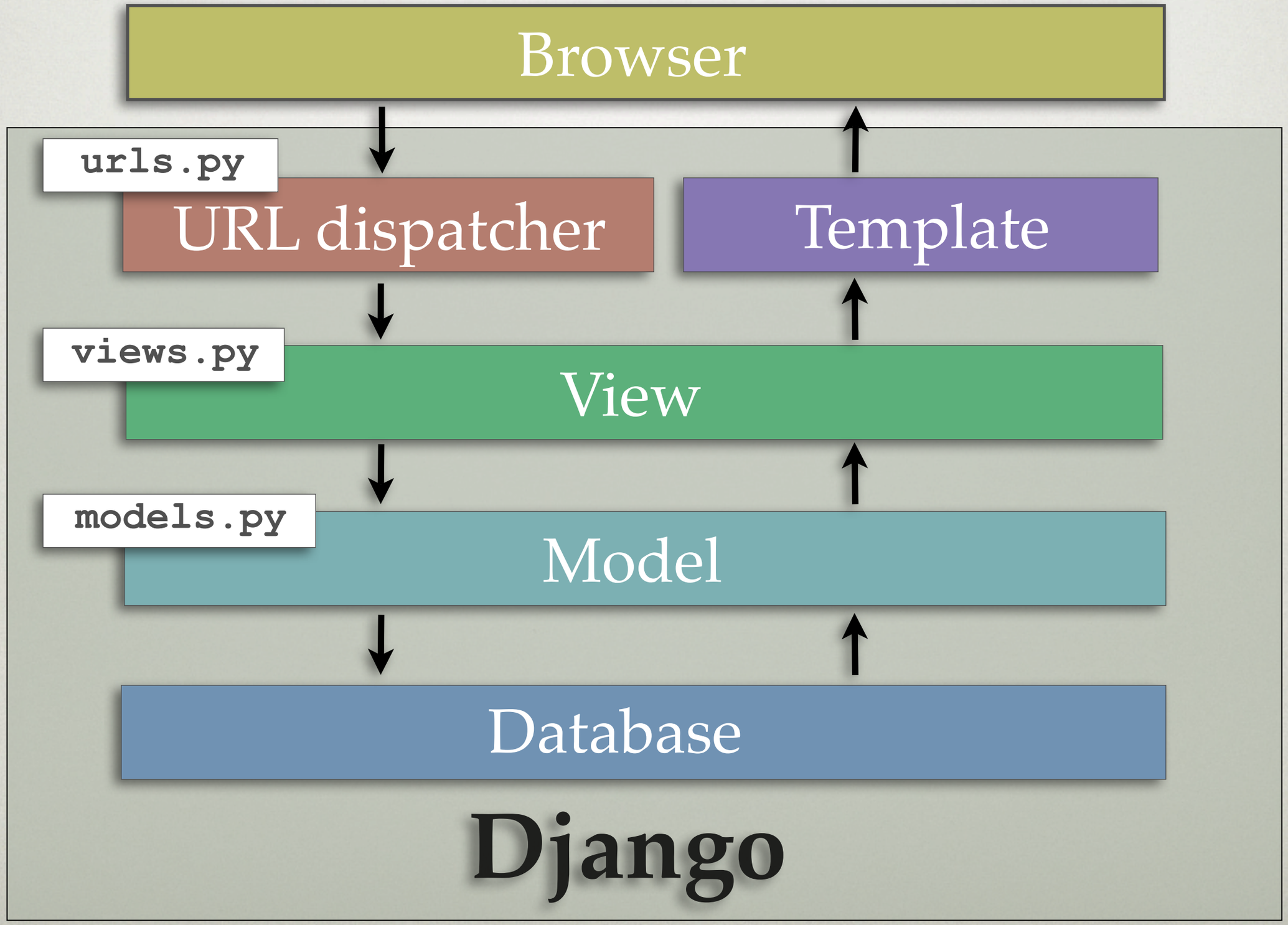
Entry 3

Published on October 22, 2011

What value can functional level test automation provide? The value that you want to achieve will depend on your objectives or reasons for automating testing. In this session, Mark explains a variety of values that have been achieved in terms of benefits and return on investment. Case studies are used ...

Read file entry

Django Architecture



Benefits of using Django

- Django projects can consist of multiple applications
  - Promotes loose coupling
    - *Applications can be developed independent of each other.*
  - Promotes tight cohesion
    - *Applications can be developed to handle specific requirements*

Benefits of using Django

- Promotes code reuse
  - *Templates can be used across applications.*
  - *Applications can be used across Django projects.*
    - *Applications just have to exist in the PYTHONPATH to be used by Django.*
- Provides several functions that handle common functions when developing web applications.

Benefits of using Django

- Django provides complete set of components that are very well documented.
- Full stack framework.
  - *Gives you everything you need to create your web app*
- Built-in Object Relational Mapper
  - *Removes the need for embedded SQL*

Benefits of using Django

- Everything is kept *"Pythonic"*
  - *Configuration files are pure Python.*
  - *URLs map to simple functions.*
  - *Database objects are just Python objects.*
- Django provides a free online book available at:
  - http://www.djangobook.com
- Promotes rapid development

Overview of Django

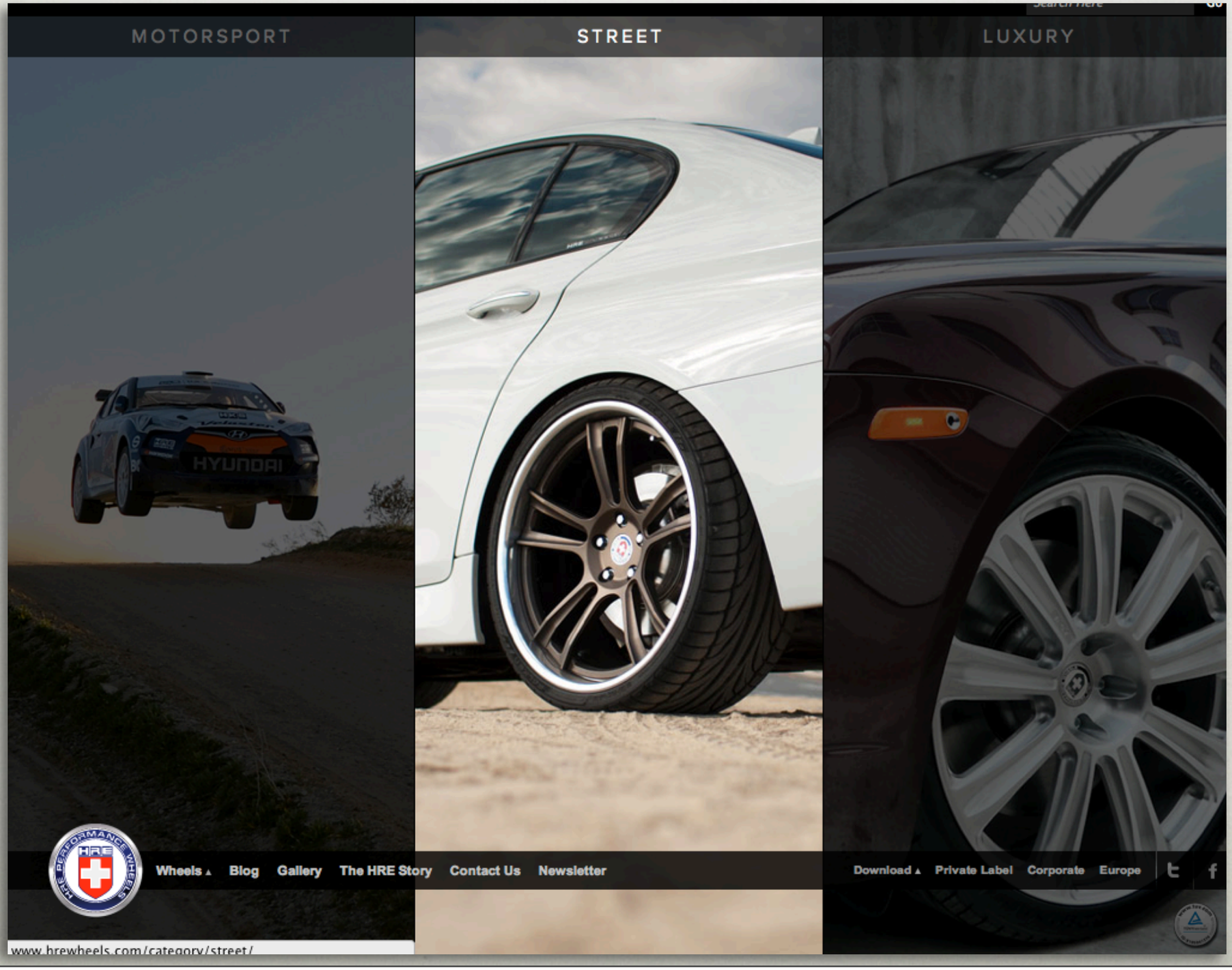
- Django uses inheritance to allows developers to take advantage of the framework's API.
  - *Custom code must override default implementation.*
- Provides support for Model, View, Controller pattern.
- Django provides a pre-built administration panel for your applications.
  - *The admin site also requires inheritance to provide administration over model objects.*

Sites that use Django

www.webcubecms.com

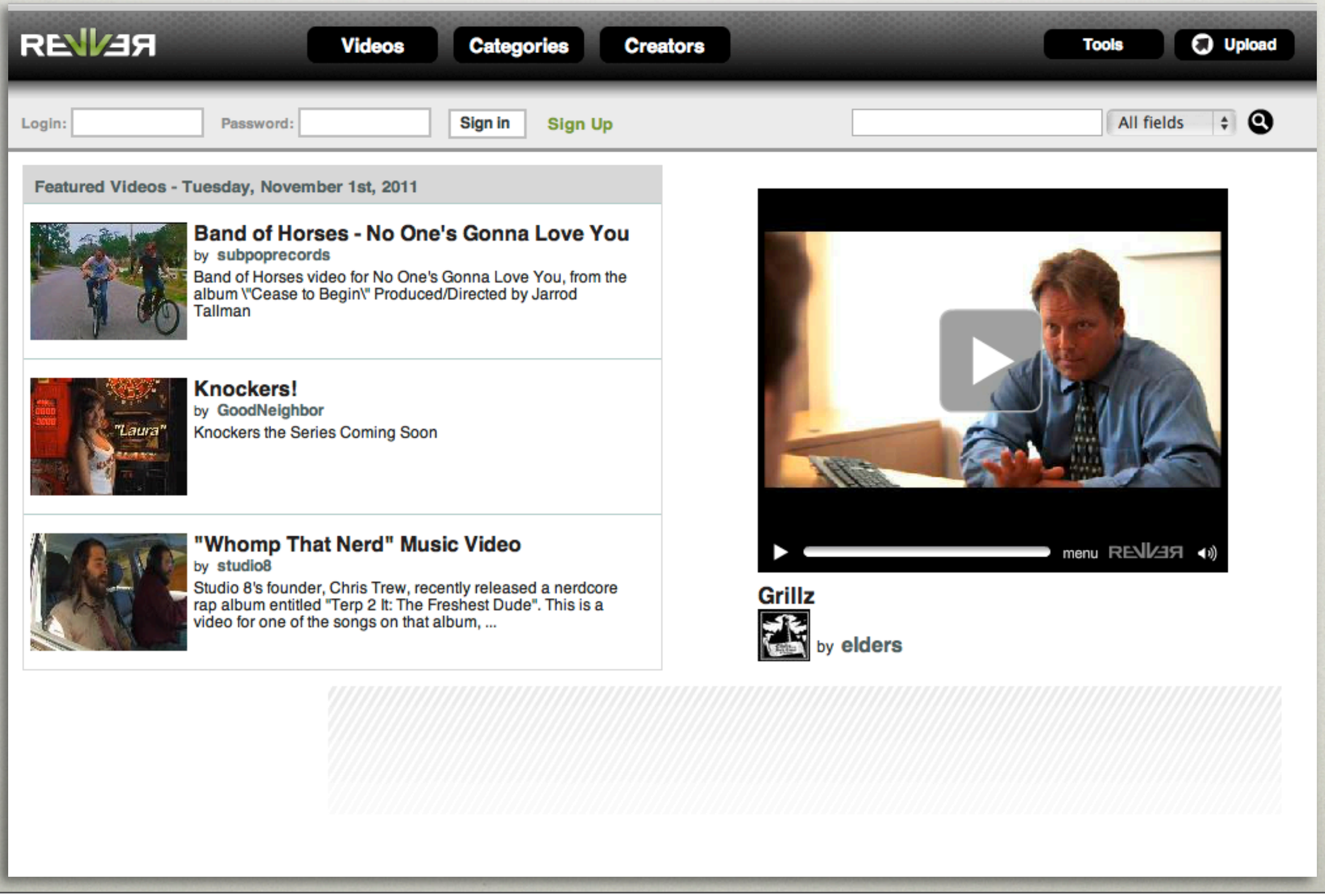
Sites that use Django

www.hrewheels.com



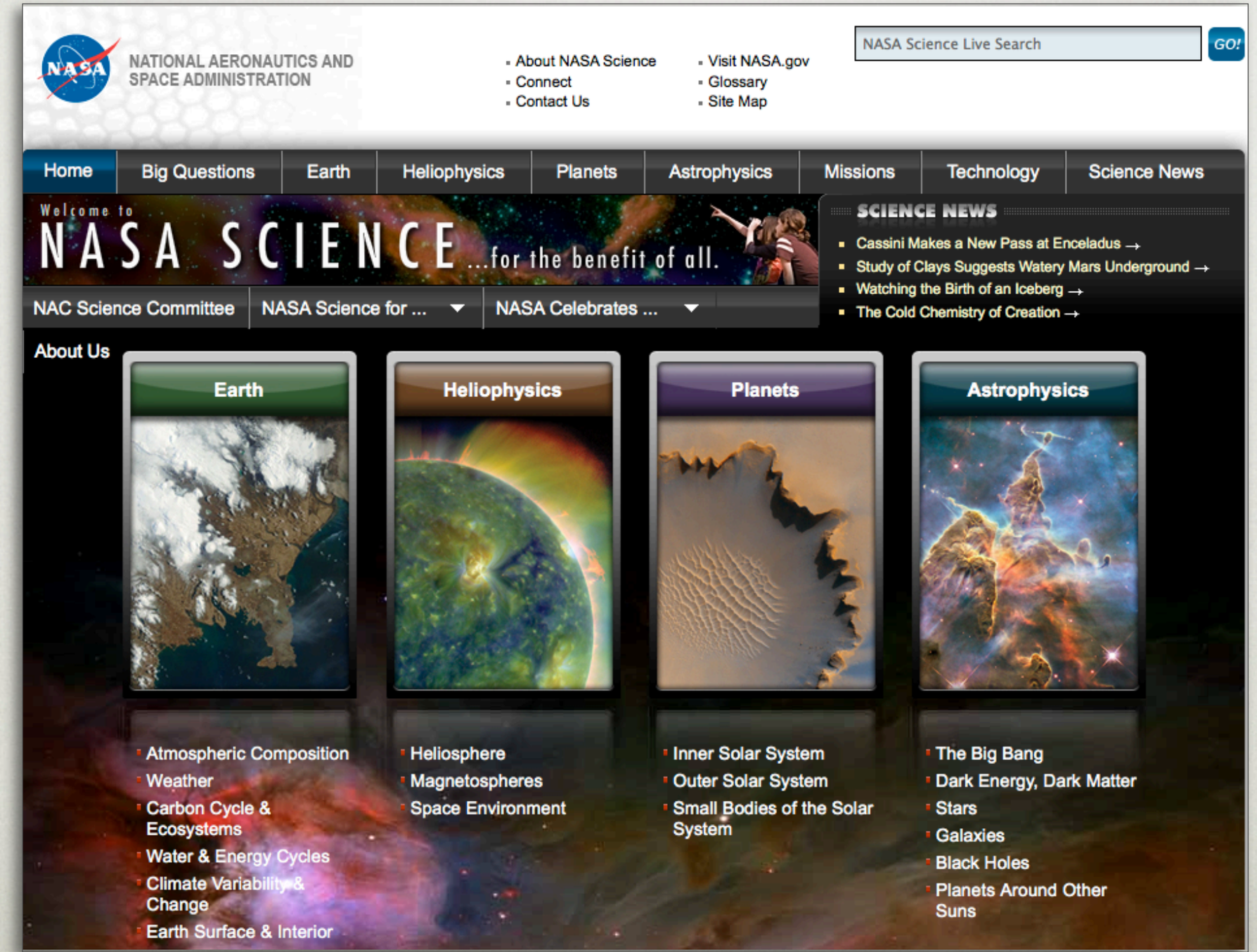
Sites that use Django

www.revver.com



Sites that use Django

http://science.nasa.gov/

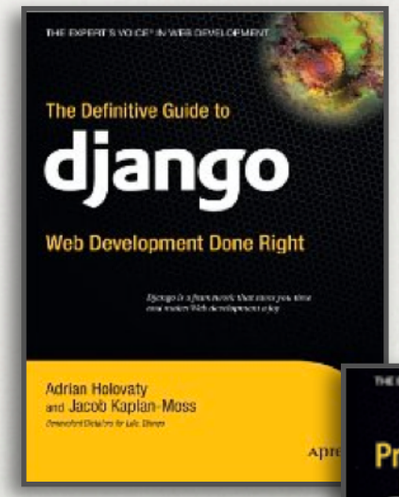


For more about Django

- www.djangoproject.com

- www.djangobook.com

  Jacob Kaplan-Moss



- Practical Django Projects, Second Edition

  James Bennett

- Django 1.0 Template Development

  Scott Newman