Django - an introduction

An introduction to Django

amckay

First off

- User group
- Is there room? Should we just go to GeekUp instead?
- Focus, speakers?
- Where should we meet?
- What do we need? Mailing lists, etc

About me

- Done lots of Plone stuff
- Now do rails and Plone stuff for Blue Fountain
- Confession: I'm a Python bigot

Django

- A web framework in the manner of the familar Ruby on Rails model
- Provides model, view, controller* around an object to relational mapper
- In other words
    - Stuff lives in a relational database
    - Simple views, models to pull it in and out
    - No need to write SQL
    - There aren't controllers in the usual sense see 1 and 2

Why Django?

- Python
- Documented
- Lots of friendly users
- Fast development and running

- "We originally grew our own framework, but have adopted Django because it's documented pretty well, and the automated admin makes certain kinds of projects economic for the first time - essentially creating a custom data model for a client and letting them administer the data. We are using it successfully on half a dozen major commercial projects. There are still a few idiosyncracies but by and large it does what it says on the tin..."

- *Andy Robinson, Reportlab creator and Python guru*

Installing

- OS: any that run Python (Windows, OS X, Linux)
- Database
  - Postgres
  - MySQL
  - SqlLite
- Web server
  - mod_python with Apache
  - standalone
  - FastCGI with Apache or lighttpd
  - many more

A first application

- Create a model
  - In a model you explicitly state what fields and where
  - Example:

  -
    ```
    class Blog(models.Model):
        title = models.CharField(blank=False, maxlength=200)
        url = models.CharField(blank=False, maxlength=200)
    ```
- Next run syncdb and it will automatically create the tables for you

Code Layout

- For an site there are multiple applications
- Allows easy reuse, all the models and views for one section are in one folder
- Configuration is through a sites settings.py file

  -
    ```
    INSTALLED_APPS = (
        'django.contrib.admin',
        [snip]...
        'blogs', )
    ```

Creating HTML

- All very exciting, but need to interact with it
- Django provides an out of the box admin interface, add in:

    -
      ```
          class Admin:
              pass
          def __str__(self):
              return self.title
      ```
- This provides you a nice way to show the blog in the admin and enables the admin
- Now go to: /admin...

Ooh

- The admin interface is not designed
    - to be the end user interface
    - it's an internal or administrator interface
    - many people spend time subverting the admin interface for their needs
    - I can see many usecases where the admin interface is enough
- Pulling together other models is easy using relationships

Posts

- Adding in posts for the blog

    -
      ```
      class Post(models.Model):
          title = models.CharField(blank=False, maxlength=200)
          post = models.TextField(blank=False)
          category = models.ForeignKey(Categories)
          blog = models.ForeignKey(Blog)
          timestamp = models.DateTimeField(blank=True,
                          auto_now_add=True)
      ```
    -

Templating

- So if you want an external interface you write a view
- A view handles a request, talks to the models and renders a template back out

- I hate the Django HTML templating language
- It looks like this:

    -
      ```
      <h1>{{ section.title }}</h1>

      {% for story in story_list %}
      <h2>
        <a href="{{ story.get_absolute_url }}">
          {{ story.headline }}
        </a>
      </h2>
      ```

A better templating language

- TAL is far better for a gazillion reasons
- Fortunately the templating is easy to change (see 1)
- With TAL this becomes:

    -
      ```
      <h1 tal:content="section/title">Title</h1>

      <tal:block repeat="story story_list">
      <h2>
        <a href="url"
           tal:attributes="url story/get_absolute_url"
           tal:content="story/headline">Story</a>
      </h2>
      ```

Views

- Grab the request and render the template eg:

    -
      ```
      def blog_detail(request, blog_url, post_id):
          blog = get_blog(blog_url)
          template = simpletemplate.get_template(
              "blogs/blog_detail.pt")
          context = {
              "object": get_object_or_404(models.Post, p
                          k=post_id),
              "margin": get_margin(blog),
              "blog": blog,
          }
          return HttpResponse(template.render(context))
      ```
    -
    -

URLs

- How does a request become a view?
- A url's module grabs the incoming URL and figures out the appropriate view
- For example:

    -
      ```
      (r'^(?P<blog_url>\w+)/(?P<post_id>\d+)/$',
                'blogs.views.blog_detail'),
      ```
- Eek regex, but let's you be really flexible
- So there we have it:
    - request > url > view > model > template

More bits and peices

- Django has many peices you need
    - Authentication
    - Users and Groups
    - Permissions and security
    - Test framework
    - Internationalisation
    - Mutliple sites
    - Easy packaging and re-use
    - File system (blob) support
- What's missing (say compared to Rails)
    - Fixtures (coming in 1.0)

Why I prefer to Rails

- Ruby is magic

    - Rails is more magic
    - Ruby on Rails is magic squared
    - I'm a muggle. I hate magic.
- Django has no magic
    - Admin interface stops me doing tedious rubbish. Scaffold is useless
    - Re-use is there and it's easier to manage
    - Python is better
    - Code seperation is clearer

Comparisons by others

Rails performed much better than Symfony. And Django performed much better than Rails.

http://wiki.rubyonrails.com/rails/pages/Framework+Performance

> While choosing between these two frameworks may be difficult, the good news is that either framework is a good choice a team wishing to develop a web application.

http://docs.google.com/View?docid=dcn8282p_1hg4sr9

> Django has won over the nearest competitors with the approximate triple superiority

http://www.alrond.com/en/2007/jan/25/performance-test-of-6-leading-frameworks/

http://wiseheartdesign.com/2006/12/6/rails-needs-something-better-than-engines/

http://www.jacobian.org/writing/2005/dec/05/ripoff/

> So, why am I back to Rails for my next project? 3 letters: FUN. I find Ruby and Rails to be pleasant to use.

http://blog.carlmercier.com/2007/01/30/why-i-moved-from-ruby-on-rails-to-pythondjango-and-back/

Questions

- Presentation will go on my blog: http://www.agmweb.ca/blog/andy
- Providing Django or TAL doesn't blow up in the meantime
- or amckay@bluefountain.com or andy@agmweb.ca