Introduction to Django Templates

May 24th, 2010

Kevin Mooney

About Me

- Kevin Mooney
- @kmooney - github.com/kmooney
- Web Cube
- http://www.webcubecms.com

Intro to Django

- The Story So Far:
  - Building a Library
    - Models - Business Objects, Logic, Persistent Storage
    - Views - URL Routing, Program Structure & Organization

Templates

- What is a Django Template?
  - Why should I...?
- Template Tags and Variables
- How to Render
- All About Context
- Flow Control
- Inheritance

What is a Django Template?

- HTML-based language. You write templates page by page.
- Templates contain presentation code, variables and simple control structures.
- Great for front-end development.

Why Should I...

- It's possible to return straight text and HTTP headers from a view, and that's how I like it! So...
- Why should I use templates anyways?
    - Templates are front-end developer & designer friendly *(team friendly)*
    - Keep your code organized

Back to the Library...

Let's Add a Home Page

- Right now, the library's root URL route gives us a 404.
- Let's create a home page.
- Code Example (library/templates/library/index.html)

library/templates/library/index.html

```
<!doctype html>
<html>
<head><title>{{ page_title }}</title></head>
<body>
    <h1>
        {{ page_title }}
    </h1>
    <div id="content">
        <ul>
            <li><a href="{% url library.views.book_index %}">Books</a></li>
            <li><a href="{% url library.views.authors %}">Authors</a></li>
        </ul>
    </div>
    <div id="copyright">&copy; 2012 AWPUG</div>
</body>
</html>
```

What was that?

- Looked like HTML, but there were % and {}

What was that?

{{ author_name }}

{% tag_name %}

What was that?

{{ author_name }} {% tag_name %}

$\uparrow$

Variable

- Double curly-brace
- Set in the *View*
- Part of *Template Context*
- You control which variables are available

What was that?

{{ author_name }}

↑

Variable

- Double curly-brace
- Set in the *View*
- Part of *Template Context*
- You control which variables are available

{% tag_name %}

↑

Template Tag

- Part of the template language
- Functions, control statements
- You can write your own, too!

Variables

- Template variables are set in the *View*
- You control what variables are available to a template by creating a *Context Object*.
- When you render a template, you pass the *Context Object* to the template renderer.

Rendering the Home Page

- The template is written, but how to serve it to a user?
- Render it in the view, then return an *HTTPResponse* object.

Rendering a Template (old-school view)

(library/views.py - index)

views.py

```
from django.template.loader import get_template
from django.template import Context

def index(request):
    t = get_template('library/index.html')
    context = Context({ "page_title": "AWPUG Library" })
    response_body = t.render(context)
    return HttpResponse(response_body)
```

urls.py

```
urlpatterns = patterns('',
    url(r'^$', 'library.views.index'),
)
```

Rendering a Template (old-school take 2)

(library/views.py - index)

views.py

```
from django.shortcuts import render_to_response

def index(request):
    return render_to_response('library/index.html',
                              {'page_title': "AWPUG Library"})
```

urls.py

```
urlpatterns = patterns('',
    url(r'^$', 'library.views.index'),
)
```

Code Example 3 - Rendering a Template (class-based view)

views.py

```
from django.views.generic.base import TemplateView

class IndexView(TemplateView):
    def get_context_data(self, **kwargs):
        return {'page_title':"AWPUG Library"}
```

urls.py

```
urlpatterns = patterns('',
    url(r'^$', IndexView.as_view(template_name='library/index.html'),
)
```

WAR?

But Wait..

Isn’t that more work than the turtle-on-a-skateboard?

Code Example 4 - Rendering a Template (class-based view, again)

views.py

Absolutely Nothing!

urls.py

```
from django.views.generic.base import TemplateView

urlpatterns = patterns('',
    url(r'^$', TemplateView.as_view(template_name="library/index.html",
                                    get_context_data=lambda:{'page_title':"AWPUG Library"}))
)
```

Advanced Variables

- Context Processors
- *RequestContext*
  - When used in concert, these can create special context variables that are available to all templates whose views use *RequestContext*
- Examples: STATIC_URL, Error Messages
- You be careful with those!

Template Tags

- Lots of Tags Built-In!
- Flow Control Tags
  - {% if %}
  - {% for %}
  - {% ifequal %}
- Inheritance Tags
  - {% extends %}
  - {% block %}

Code Example 4 - Flow Control

(templates/library/author_list.html)

templates/library/author_list.html

```
<html>
<head>
    <title>List of all authors</title>
</head>
<body>
    <h1>All the authors we know about:</h1>
    <ul>
        {% for author in object_list %}
            <li>{{ author.name }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

views.py

Nothing!

urls.py

```
urlpatterns = patterns('',
    url(r'^authors/$', ListView.as_view(model=Author), name='library.views.authors'),
)
```

Orlando Magic

List View Magic

- You don’t need to say which template to use.
- Just name it <<model>>_list.html
- ListView knows the model, so it can figure out a good Queryset and a good template variable name. (80-20 rule)
- Just remember the naming convention and ListView does the rest for you!

Back To The Library...

- Now we have a sweet home page with a header and a copyright statement at the bottom.
- Wouldn’t it be cool to have that stuff on all the pages?
- But sooo muchh work....

Back To The Library...

- Now we have a sweet home page with a header and a copyright statement at the bottom.
- Wouldn't it be cool to have that stuff on all the pages?
- But sooo muchh work....

Template Inheritance to the Rescue!

- Use {% extends %} and {% block %}
- Write your headers and footers and get your page structure figured out just once.
- Then apply it everywhere!

Code Example 5 - Template Inheritance

Plan of Attack

- Use {% extends %} and {% block %} tags to move common elements to base.html
- Establish blocks that can be overridden if needed.
- Create child templates.

library/templates/library/base.html

```
<!doctype html>
<html>
<head><title>{{ page_title }}</title></head>
<body>
    <h1>
    {% if page_title %}
        {{ page_title }}
    {% else %}
        Untitled Page
    {% endif %}
    </h1>
    <div id="content">
    {% block content %}
    {% endblock %}
    </div>
    <div id="copyright">&copy; 2012 AWPUG</div>
</body>
</html>
```

library/templates/library/index.html

```
{% extends 'library/base.html' %}

{% block content %}
<ul>
    <li><a href="{% url library.views.book_index %}">Books</a></li>
    <li><a href="{% url library.views.authors %}">Authors</a></li>
</ul>
{% endblock %}
```

library/templates/library/author_list.html

```
{{% extends 'library/base.html' %}

{% block content %}
<h2>All the authors we know about:</h2>
<ul>
    {% for author in object_list %}
        <li>{{ author.name }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

Overview

- Child templates start with {% extends %} to show that they inherit from a template.
- The parent template uses {% block %} to indicate what portions of the template can be overridden.
- Everything that is outside of a {% block %} cannot be changed by children.

Questions?

- Kevin Mooney
- github.com/kmooney
- @kmooney