TemplatesandViews

January 7, 2019

1 Django Templates

Template mechanism provides the division of responsibilities on the web pages generated. The content/data, its HTML representation and application logic is maintained on different parts. Control loads a template file with a context and template engines generates the HTML (or other format) pages from context.

A template is usually a HTML file with some special template markups. Django templates use: * `{{ variablename }}` to substitute a variable from the context. * `{% block .... %} Block content in multiple lines. If necessary {% endblock %}` to end block expressions.

The `{{ variable }}` syntax support indexing, key selection, attribute selection and even function calls without arguments as. `{{var.name}}` will try: * if `name` is a number like 0, `var[0]` * `var['name']` * `var.name` * if `var.name` is a callable, `var.name()`.

Blocks can contain plain HTML code directly copied and other blocks or variable expansions. The following is some of the block expressions supported: * `{# commented text here #}` Used for line-in comments. * `{% comment %} ...... {% endcomment %}` Multi line, block comments. * `{% if boolexrp %}` ... `{% elif boolexpr %}` ... `{% endif %}`. For conditional expansion of enclosed template parts. `boolexpr` can contain variable references, `and`, `or`, `==`, `!=`, `in`, `not in`, and comparison operators. * `{% for var in variable %}` ... `{% endfor%}`. Repeats the enclosed template part for all elements in the iterated variable. Used in table, list or lines generated from a data structure. An optional `{% empty %}` part before `endfor` defines if iterator returns no result. * `{% now "D d/m/Y h:i" %}` displays current time specified in format string * `{% block blockname %}` ... `{% endblock %}` defines a named block to be used in template inheritence * `{% extends "templatepath" %}` defines a base template. Base template is loaded and the named blocks are replaced by the named blocks defined in current template * `{% include "templatepath" %}` another template is included * `{% filter filtername[|filtername]* %} ... {% endfilter %}` A filter is applied in the block. See filters below. * `{% spaceless %}` ... `{% endspaceless %}` template part has all space between tags removed. A more compact HTML is generated. * `{% csrf_token %}` generates a Cross-Site Request Forgery prevention token in template output. See https://docs.djangoproject.com/en/2.0/ref/csrf/

The filters can be used to filter template output. They are functions that map values. A filter can be applied by `{{ value|filter }}` . Filters can be used to convert output into uppercase, lowercase, change format, escape for HTML, javasript, etc. They can be applied by `{% filter ... %}` tag to complete blocks as well.

See https://docs.djangoproject.com/en/2.0/ref/templates/builtins/ for a complete list of tags and templates

1.1 Template Inheritence

Django templates can use inheritence to modularly define multiple page layouts. A template extending another replaces only the defined blocks as:

```
{# this is base.html #}
<html><body>
    <div>{% block 'sidebar'}{% endblock %}</div>
    <div> <h1>Wellcome to My Application </h1>
          {% block 'content'}{% endblock %}
    </div>
    </body></html>
```

Then another extends this:

```
{# this is studentview.html #}
{% extends 'base.html' %}
{% block 'sidebar' %}
<a href="#sect1">Section 1</a>
<a href="#sect2">Section 2</a>
{% endblock %}

{% block 'content' %}
Here is the student list:<br/>
<ul>
{% for s in student %}
    <li>{{s.name}}, {{s.surname}}. id: {{ s.sid }}</li>
{% endfor}
</ul>
{% endblock %}
```

This way changing main document layout will not affect the content templates. Also code repetitions will be avoided. Generated HTML for `studentview.html` will be:

```
<html><body>
    <div><a href="#sect1">Section 1</a>
<a href="#sect2">Section 2</a></div>
    <div> <h1>Wellcome to My Application </h1>
     Here is the student list:<br/>
<ul>
    <li>Onur, ehitolu. id: 55727</li>
    <li>Cin, Ali. id: 55571</li>
    <li>Nasrettin, Hoca. id: 59213</li>
</ul>
    </div>
    </body></html>
```

1.2 Using Templates in the Views

You can use functions in `django.template.loader`, `get_template()` and `select_template()`. They return `Template` objects and you can use `render()` method. A shorter version is to use `django.shortcut.render`:

```
from django.shortcuts import render

def index(request):
    context = { 'student' : [{'name':'Onur', 'surname':'ehitolu', 'sid':55727}] }
    render(request, 'studentview.html', context)
```

The template files are searched under directories specified in `settings.py` for the project. Also `templates` directory under the application directory is looked up.

2 Processing User Data in Views

`HTTPRequest` objects passed as the first parameter of a view contains all information related to request: * `method`: HTTP method `GET`, `POST`, `PUT`, etc. * `path`: path part of the url http://example.com/app/test#section URL path is `/app/test#section` * `META`: a dictionary like object to get most HTTP request parameters * `POST`: a dictionary of posted form data * `GET`: a dictionary of passed form data through `GET` method. * `COOKIES`: a dictionary of cookie variables set * `FILES`: a dictionary of uploaded files (with `<INPUT TYPE="FILE" ...>` tag. `POST` does not contain this information * `body`: raw HTML request body (it is possible to send XML or other mime-types)

```
from django.http import HttpResponse
from student.models import Student

def index(request):
    formdata = if request.method == 'GET' request.GET else request.POST
    sid = formdata['sid']
    name = formdata['name']      # assume they are posted
    surname = formdata['surname']
    Student.objects.create(sid, name,surname)
    return HTTPResponse('<html><body>new stududent successfully created')
```

3 Forwarding URL into View

Either directly map or incude applications `urls.py` in projects `urls.py`

```
"""
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/1.10/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
Class-based views
```

```
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.conf.urls import url, include
    2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
"""
from django.conf.urls import url,include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    # all urls starting with student/ is searched in student/urls.py
    url(r'^student/', include('student.urls')),
]
```

In applications `urls.py` you can match the remainder of the URL:

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url('^$', views.index, name='index'),
    url('^detail/(?P<stid>[0-9]{5,})?$', views.detail, name='detail'),
    url('^update/(?P<stid>[0-9]{5,})$', views.update, name='update'),
    url('^add$', views.add, name='add'),
    url('^register/(?P<stid>[0-9]{5,})?$', views.register, name='register'),
]
```

The named groups in regular expressions are passed to view functions as keyword arguments.

3.1

Request/Response Path

1. browser makes a request to django server/web server for a URL
2. URL is matched in `urls.py` and included views. match returns a view function/class.
3. An HTTPRequest object is constructed by django containing, session, POST, GET, user, COOKIE, FILES and other request context. Matched view is called with this context and optional keyword arguments in url match.
4. Python code in view function creates an `HttpResponse` object and returns it.
5. Django server converts this HttpResponse object into an HTTP response and sends to the browser

`HttpResponse` object can be created in different ways: * python from django.http import HttpResponse .... return HttpResponse(contenttext, content_type='text/html') * python from django.shortcuts import render ... render(request, templatepath) Which will return a `HttpResponse` object. * python res = HttpResponse() res['Content-type'] = 'text/html' ... incrementally update `response    res.write("hello world")    return res` * You can use `StreamingHttpResponse` and `FileHttpResponse` objects to send larger response bodies. Server will send data to browser as you write data to this stream and server will read file and write to browser respectively

3.2 Adding a View

A typical operations to add a new view (page) 1. Create a template for the view in `tmplates/newpage.html` 1. Implement the view in `views.py` as `def newpage(request):..` 1. Add a matching url entry in `urls.py`