# Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

## Lecture topic

URL mapping in Django

# URL mapping in Django: Table of content

# Introduction

# URL dispatcher

- A clean, elegant URL scheme is an important detail in a high-quality Web application
- Django lets you design URLs however you want, with no framework limitations.
- There's no .php or .cgi required, and certainly none of that 0,2097,1-1-1928,00 nonsense (like on www.wp.pl, for example)
- *Cool URIs don't change* by World Wide Web creator Tim Berners-Lee http://www.w3.org/Provider/Style/URI

# How Django processes a request

- Determine the root URLconf module - the ROOT_URLCONF setting or the `urlconf` attribute of the incoming HttpRequest object (default: `urls.py` in the Django project directory)
- Load the root URLconf module and find the `urlpatterns` variable
- Find the first URL pattern that matches the requested URL. Request method (GET, POST etc. ) and parameters are ignored (but passed to the view function later on)
- Import and call the view function assigned to the matching URL pattern

# URLconf modules

# Sample URLconf

```
1  from django.conf.urls.defaults import *
2
3  urlpatterns = patterns('',
4      (r'^articles/2003/$', 'news.views.special_case_2003'),
5      (r'^articles/(\d{4})/$', 'news.views.year_archive'),
6      (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
7      (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
8  )
```

Important notes:

- the first import makes the `patterns()` function available
- To capture a value from the URL, just put parenthesis around it
- No need to add a leading slash (`âarticles`, not `ˆ/articles`)
- The `'r'` tells Python that a string is “raw”, i.e., nothing in it should be escaped.
- The order of URL patterns is important!

# Sample URLconf

```
1  from django.conf.urls.defaults import *
2
3  urlpatterns = patterns('',
4      (r'^articles/2003/$', 'news.views.special_case_2003'),
5      (r'^articles/(\d{4})/$', 'news.views.year_archive'),
6      (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
7      (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
8  )
```

- A request to `/articles/2005/03/` would match the third entry in the list. Django would call the function `news.views.month_archive(request, '2005', '03')`
- `/articles/2005/3/` would not match any URL patterns (the third entry in the list requires two digits for the month)
- `/articles/2003/03/3/` would match the final pattern. Django would call the function `news.views.article_detail(request, '2003', '03', '3')`

# Named groups

```
1  from django.conf.urls.defaults import *
2
3  urlpatterns = patterns('',
4      (r'^articles/2003/$', 'news.views.special_case_2003'),
5      (r'^articles/(?P<year>\d{4})/$', 'news.views.year_archive'),
6      (r'^articles/(?P<year>\d{4})/(?P<month>\d{2})/$',
7       'news.views.month_archive'),
8      (r'^articles/(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d+)/$',
9       'news.views.article_detail'),
10 )
```

- The captured values are now passed to view function as keyword arguments, rather than positional arguments
- A request to `/articles/2005/03/` would call the function `news.views.month_archive(request, year='2005', month='03')`
- If there are any named arguments, non-named arguments will be ignored
- Advice: do not mix named and non-named groups

# The `patterns()` function

`patterns(prefix, pattern_description, ...)`

- Each `pattern_description` should be in the following format:

```
1  (regular expression,
2   Python callback function
3   [, optional dictionary [, optional name]])
```

- `patterns()` returns a Python list, so the following is perfectly ok:

```
1  urlpatterns = patterns('', ...)
2  urlpatterns += patterns('', ...)
```

- Use `prefix` to cut down on code duplication

```
1  urlpatterns = patterns('mysite.news.views',
2      (r'^articles/(\d{4})/$', 'year_archive'),
3      (r'^articles/(\d{4})/(\d{2})/$', 'month_archive'),
4      (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'article_detail'),
5  )
```

# Specifying the view function in pattern descriptions

The callback view function can be specified as a path to the module...

```
1  urlpatterns = patterns('',
2      (r'^archive/$', 'mysite.views.archive'),
3      (r'^about/$', 'mysite.views.about'),
4      (r'^contact/$', 'mysite.views.contact'),
5  )
```

or as a Python object (function) itself:

```
1  from mysite.views import archive, about, contact
2
3  urlpatterns = patterns('',
4      (r'^archive/$', archive),
5      (r'^about/$', about),
6      (r'^contact/$', contact),
7  )
```

The two examples above achieve exactly the same goal - which one you use is etirely up to you!

# Optional parameters in pattern descriptions

```
1  (regular expression,
2   Python callback function
3   [, optional dictionary [, optional name]])
```

- `optional dictionary` may contain extra keyword arguments that will be passed to the view function.

```
1  urlpatterns = patterns('blog.views',
2      (r'^blog/(?P<year>\d{4})/$', 'year_archive', {'foo': 'bar'}),
3  )
```

- `optional name` allows to reverse URLs, i.e., to retrieve view functions associated with url patterns (will be explained later on). The `url` function allows to skip extra keyword arguments:

```
1  urlpatterns = patterns('',
2      url(r'^archive/(\d{4})/$', archive, name="full-archive"),
3      url(r'^archive-summary/(\d{4})/$', archive,
4          {'summary': True}, "arch-summary"),
5  )
```

# The `include()` function

`include(<module or pattern_list>)`

```
1  from django.conf.urls.defaults import *
2
3  extra_patterns = patterns('',
4      url(r'reports/(?P<id>\d+)/$', 'credit.views.report',
5          name='credit-reports'),
6      url(r'charge/$', 'credit.views.charge',
7          name='credit-charge'),
8  )
9
10 urlpatterns = patterns('',
11     url(r'^$',     'apps.main.views.homepage', name='site-homepage'),
12     (r'^help/',    include('apps.help.urls')),
13     (r'^credit/', include(extra_patterns)),
14 )
```

- `include()` essentially "roots" a set of URLs below other ones
- Use single quotation marks to include URLconf modules
- Skip single quotation marks to include lists returned by `patterns()`

# `include()` and captured parameters

An included URLconf receives any captured parameters from parent URLconfs:

```
1   # In settings/urls/main.py
2   urlpatterns = patterns('',
3       (r'^(?P<username>\w+)/blog/', include('foo.urls.blog')),
4   )
5
6   # In foo/urls/blog.py
7   urlpatterns = patterns('foo.views',
8       (r'^$', 'blog.index'),
9       (r'^archive/$', 'blog.archive'),
10  )
```

- Both `blog.index` and `blog.archive` will receive a keywoard parameter `username`
- Be careful not to mix named and unnamed groups (unless intended)

# URL namespaces

- URL namespaces allow to differentiate between different instances of the same application
- An URL namespace comes in two parts:
  - An application namespace describes the name of the application that is being deployed.
  - An instance namespace identifies a specific instance of an application

Two ways to speicify URL namespaces:

- Provide arguments to `include()`:

```
1   (r'^help/', include('apps.help.urls',
2    namespace='foo', app_name='bar')),
```

- Include a 3-tuple instead of a just a list returned by `patterns()`:

```
1   (<patterns object>, <application namespace>, <instance namespace>)
```

# URL reversing

# Basics of URL reversing

Consider the following urlpatterns:

```
1  urlpatterns = patterns('',
2      url(r'^archive/(\d{4})/$', archive, name="full-archive"),
3      url(r'^archive-summary/(\d{4})/$', archive,
4          {'summary': True}, "arch-summary"),
5  )
```

It is convenient to be able to retrieve the URL leading to a given view function. You can target each pattern individually by using its name:

- In templates: using the `url` template tag:

```
1  {% url arch-summary 1945 %}
2  {% url full-archive 2007 %}
```

- In Python code: using the `reverse()` function:

```
1  from django.core.urlresolvers import reverse
2
3  def myview(request):
4      return HttpResponseRedirect(reverse('arch-summary', args=[1945]))
```

# More on the `reverse()` function

```
reverse(viewname, urlconf=None, args=None,
kwargs=None, current_app=None)
```

- `viewname` is the name of the view function to be reversed
- `urlconf` can be normally omitted
- `args` are positional arguments to the view function
- `kwargs` are keyword arguments to the view function
- `current_app` allows to specify the app that the currently executing view belogs to

Important notes:

- Currently, patterns containing the "|" character cannot be reversed
- You can provide either `args` or `kwargs`, never both

# Reversing namespaced URLs

- Namespaced URLs are specified using the : operator. For example, the main index page of the admin application is referenced using `admin:index`
- Namespaces can also be nested. The named URL `foo:bar:whiz` would look for a pattern named whiz in the namespace bar that is itself defined within the top-level namespace foo.
- The detailed strategy of reversing namespaced URLs is described in the Django documentation.

# Other useful utility functions

# the `get_absolute_url()` method

`Model.get_absolute_url()`

- Define a `get_absolute_url()` method to tell Django how to calculate the URL for an object. For example:

```
1   def get_absolute_url(self):
2       return "/people/%i/" % self.id
```

- If an object defines this method, the object-editing page in the admin interface will have a "View on site" link that will jump you directly to the object's public view
- It's good practice to use `get_absolute_url()` in templates, instead of hard-coding your objects' URLs. For example, this template code is bad:

```
1   <a href="/people/{{ object.id }}/">{{ object.name }}</a>
```

But this is ok:

```
1   <a href="{{ object.get_absolute_url }}">{{ object.name }}</a>
```

# the `permalink()` decorator

Problem: the way we wrote `get_absolute_url()` above is that it slightly violates the DRY principle: the URL for this object is defined both in the URLconf file and in the model. Solution: use the `permalink()` decorator:

If your URLconf contained a line such as:

```
1  (r'^people/(\d+)/$', 'people.views.details'),
```

. . . your model could have a `get_absolute_url()` method that looked like this:

```
1  from django.db import models
2
3  @models.permalink
4  def get_absolute_url(self):
5      return ('people.views.details', [str(self.id)])
```

Note: We could also assign a name to the view function and return it in the "permalinked" `get_absolute_url()` function.

# the `resolve()` function

`resolve(path, urlconf=None)`

Can be used for resolving URL paths to the corresponding view functions - the opposite of `reverse()`

- `path` is the URL path you want to resolve
- `urlconf` can be omitted, as in `reverse()`
- returns the triple (view function, arguments, keyword arguments)
- Can be used for testing if a view would raise a `Http404` error:

# `resolve()` example

```
1  from urlparse import urlparse
2  from django.core.urlresolvers import resolve
3  from django.http import HttpResponseRedirect, Http404
4
5  def myview(request):
6      next = request.META.get('HTTP_REFERER', None) or '/'
7      response = HttpResponseRedirect(next)
8      # modify the request and response as required, e.g. change locale
9      # and set corresponding locale cookie
10     view, args, kwargs = resolve(urlparse(next)[2])
11     kwargs['request'] = request
12     try:
13         view(*args, **kwargs)
14     except Http404:
15         return HttpResponseRedirect('/')
16     return response
```

# Sources

Django documentation: www.djangoproject.com