30 December 2019

Django URLS

Django looks first in the ROOT_URLCONF declared in the settings.py file which points to the path where we need to access the urls.

In the file, we must have urlpatterns variable. This should be a sequence of path or re_path instances.

Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL.

To capture a value from the URL, use angle brackets.

```
path('articles/<int:year>/', views.year_archive),
```

Int allows us to be more specific with our value. This is known as converter type. If a converter isn't included, a string is passed.

The following path converters are available by default:

- **str** - Matches any non-empty string, excluding the path separator, **'/'**. This is the default if a converter isn't included in the expression.
- **int** - Matches zero or any positive integer. Returns an *int*.
- **slug** - Matches any slug string consisting of ASCII letters or numbers, plus the hyphen and underscore characters. For example, **building-your-1st-django-site**.

```
from django.urls import path

from . import views

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>/', views.year_archive),
    path('articles/<int:year>/<int:month>/', views.month_archive),
    path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),
]
```

Django also allows us to use regex in the path name in the form of re_path

```
from django.urls import path, re_path

from . import views

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    re_path(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$',
views.month_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<slug>[\w-]+)/$',
views.article_detail),
]

# URLconf
from django.urls import path

from . import views

urlpatterns = [
    path('blog/', views.page),
    path('blog/page<int:num>/', views.page),
]

# View (in blog/views.py)
def page(request, num=1):
    # Output the appropriate page of blog entries, according to num.
    ...
```

Including other URL Configs

```
from django.urls import include, path

urlpatterns = [
    # ... snip ...
    path('community/', include('aggregator.urls')),
    path('contact/', include('contact.urls')),
    # ... snip ...
]

from django.urls import path
from . import views

urlpatterns = [
    path('blog/<int:year>/', views.year_archive, {'foo': 'bar'}),
]
```

Reverse Resolution of URLS

Reversing an URL allows us to get the absolute url instead of hardcoding it int our app.

Django provides tools for performing URL reversing that match the different layers where URLs are needed:

- In templates: Using the **url** template tag.
- In Python code: Using the **reverse()** function.
- In higher level code related to handling of URLs of Django model instances: The **get_absolute_url()** method.

```
from django.urls import path

from . import views

urlpatterns = [
    #...
    path('articles/<int:year>/', views.year_archive, name='news-year-archive'),
    #...
]
```

USE OF REVERSE IN TEMPLATES

```
<a href="{% url 'news-year-archive' 2012 %}">2012 Archive</a>
{# Or with the year in a template context variable: #}
<ul>
{% for yearvar in year_list %}
<li><a href="{% url 'news-year-archive' yearvar %}">{{ yearvar }} Archive</a></li>
{% endfor %}
</ul>
```

USE OF REVERSE IN VIEWS

```
from django.http import HttpResponseRedirect
from django.urls import reverse

def redirect_to_year(request):
    # ...
    year = 2006
    # ...
    return HttpResponseRedirect(reverse('news-year-archive', args=(year,)))
```

While using urls reversing it is a good practice to have namespaces. This allows different apps to have the same url names without any conflict.

app_name = “APP NAME” in urls.py of the app