django$_{a}uto_{u}rlDocumentation$

Release 0.0.1

Thomas Hartmann

Sep 07, 2022

Contents:

django_auto_url is a Django app that liberates you from maintaining the `urls.py` files.

The principle is that a website (or a webapp) basically consists of a bunch of `View` and the user navigates from one `View` to the next or gets directed to `View` after some interaction.

So why the hassle with manually creating URL patterns and names?

django_auto_url provides functions and mixins to automate this process.

CHAPTER 1

Reference

1.1 Functions

django_auto_url.urls.urls.**get_urls_from_module**(*module*)

Scan a package and return the urlpatterns.

This function scans the module for all subclasses of *AutoUrlMixin*, generates a URLResolver for each of them and returns a list that you can feed directly to urlpatterns.

Parameters **module** (*module*) – The module to scan.

Returns list of URLResolver

Example

Assuming, you have your views defined in the module my_app.views, this is, how your urls.py would look like:

```
>>> from my_app import views
>>> from django_auto_url.urls import get_urls_from_module
>>>
>>> urlpatterns = get_urls_from_module(views)
```

django_auto_url.urls.urls.**reverse_classname**(*class_name*, *return_url_name=False*, *args=None*, *kwargs=None*)

Get the URL for a View.

Converts a View into its URL or the django url name.

Parameters

- **class_name** (str or subclass of *AutoUrlMixin*) – The View to reverse. You can either provide the class directly or a string with the full module path.
- **return_url_name** (*bool, optional*) – If set to True, return the django url name and not the final URL.

- **args** (*list, optional*) – The positional arguments for the view.
- **kwargs** (*dict, optional*) – The keyword arguments for the view.

Returns *str* – The URL or the django url name.

`django_auto_url.urls.urls.`**`reverse_classname_lazy`**(*class_name*, *return_url_name=False*, *args=None*, *kwargs=None*)

Lazy version of *`reverse_classname()`*.

Instead of doing the reverse instantly, it is done, when the result is actually needed. This will help you when you need to specify a URL in a class attribute. The non-lazy version would instantly try to find the URL for the view, even if the view has not been declared yet (for instance, because it is declared later in the same file or in a different one).

Arguments and return values are exactly the same as *`reverse_classname()`*.

`django_auto_url.urls.urls.`**`reverse_local_classname`**(*class_name*, *return_url_name=False*, *args=None*, *kwargs=None*, *lazy=True*)

Get the URL of a view declared in the same module.

Basically does the same as *`reverse_classname()`* but you do not need to specify the full module path to the view. The class name is enough.

Parameters

- **class_name** (str or subclass of *`AutoUrlMixin`*) – The `View` to reverse. You can either provide the class directly or a string with the full module path.
- **return_url_name** (*bool, optional*) – If set to `True`, return the django url name and not the final URL.
- **args** (*list, optional*) – The positional arguments for the view.
- **kwargs** (*dict, optional*) – The keyword arguments for the view.
- **lazy** (*bool, optional*) – Determines whether the evaluation should be lazy.

Returns *str* – The URL or the django url name.

1.2 Template Tags

`url_from_class`

Return the absolute URL for a class, providing the full module path.

This is the equivalent to the url template tag of Django.

The only important difference is, that you can only use keyword arguments!

Example

```
{% load auto_url %}

<a href="{% url_from_class "my_app.views.MyView" bool_arg="False" say_this="New␣
↪String" my_age="32" %}">Click me!</a>
```

Parameters

- **viewname** (*str*) – The full module path to the view.

- ****kwargs** (*any*) – Keyword arguments for the view.

Returns *str* – The absolute URL.

1.3 Mixins

class django_auto_url.mixins.mixins.**AutoUrlMixin**(**args*, ***kwargs*)

Mixin for automatic URL url creation.

Include this Mixin in your `View` to have a urlpattern automatically generated when this surrounding package is scanned with `get_urls_from_module()`.

Variables

- **url_path_name** (*str, optional*) – If set, this is the string that appears in the actual URL. Otherwise the class name is used.
- **url_django_name** (*str, optional*) – If set, this is the name of the route in django. Otherwise the class name is used.
- **is_index** (`bool`) – Set to `True` if this is an index view.
- **url_kwargs** (list of `kwarg`) – The list of Keyword Arguments for this view.
- **url_ignore_pk** (`bool`) – If the `View` includes a `slug_url_kwarg` or `pk_url_kwarg` class attribute (like the `DetailView`), a keyword argument for the Primary Key or the Slug is included automatically. Set this to `False` if this is not what you want.

classmethod get_url()

Generate and return the :class:‘~django.urls.resolvers.URLResolver‘s.

Returns list of `URLResolver`

classmethod get_url_django_name()

Return the django url name.

classmethod get_url_kwargs()

Return the Keyword Arguments.

classmethod get_url_path_name()

Return the url_path_name.

1.4 Keyword Arguments

Keyword arguments can be specified by using the `url_kwargs` class variable of the view. You can specify a default value, but in this case, the order matters: You cannot define a Keyword argument without a default after one for which you have specified one.

1.4.1 Example

```
class MyView(AutoUrlMixin, TemplateView):
    url_kwargs = [
                    kwargs.String('my_string'),
                    kwargs.Int('my_int', 42)
                ]
```

class django_auto_url.kwargs.kwargs.**string**(*name*, *default=None*)
String Keyword Argument.

class django_auto_url.kwargs.kwargs.**int**(*name*, *default=None*)
Int Keyword Argument.

class django_auto_url.kwargs.kwargs.**bool**(*name*, *default=None*)
Bool Keyword Argument.

CHAPTER 2

Quickstart

2.1 Install

2.2 Configure

Next, include `django_auto_url` in the `INSTALLED_APPS` section of the Django `settings.py`:

```
INSTALLED_APPS = [
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_auto_url'
    ]
```

2.3 Prepare the views

Now include the *AutoUrlMixin* in the mixins of your view class:

> **Attention:** Always include mixins **before** the view class!

```
from django.views.generic import TemplateView
from django_auto_url.mixins import AutoUrlMixin

class MyView(AutoUrlMixin, TemplateView):
    template_name = 'my_template.html'
```

2.4 Prepare urls.py

Now we need to generate the `urlpatterns` for our view in the `urls.py` file.

The `urls.py` file would look like this:

```
from my_app import views
from django_auto_url.urls import get_urls_from_module

urlpatterns = get_urls_from_module(views)
```

2.5 Get the URL for the view

django_auto_url provides several methods to get the URL of a view.

2.5.1 Use the template tag

Getting the absolute URL for a view in Django is done using the url template tag. django_auto_url provides a similar template tag, called *`url_from_class`* that performs the same task but takes the full module path to a view as the argument.

Let's consider that the view class we created above lives in the `views` package of your app called `my_app`.

Here is how to create a link to it from a template:

```
{% load auto_url %}

<a href="{% url_from_class "my_app.views.MyView" %}">Click me!</a>
```

2.5.2 Use the Python functions

In order to get the URL for a view, use one of the following functions:

- *`reverse_classname()`*
- *`reverse_classname_lazy()`*
- *`reverse_local_classname()`*

All these function are quite similar in how they work: You provide the view class or the full module path as a string and you get the URL returned.

```
from my_app import views
from django_auto_url.urls import reverse_classname

# resolve now. views.MyView must have already been declared.
url_for_view = reverse_classname(views.MyView)
url_for_view = reverse_classname('my_app.views.MyView')

# resolve later. views.MyView can be declared later.
# This is very useful if you need to provide a URL as a class variable.
url_for_view = reverse_classname_lazy(views.MyView)
url_for_view = reverse_classname_lazy('my_app.views.MyView')
```

For further details, please refer to the *appropriate section in the reference*.

CHAPTER 3

Use Arguments for the View

As usual, views can accept arguments by their URL patterns. You can do this for your *AutoUrlMixin* views by specifying `url_kwargs` like this:

```
from django.views.generic import TemplateView
from django_auto_url.mixins import AutoUrlMixin
from django_auto_url import kwargs

class MyView(AutoUrlMixin, TemplateView):
    template_name = 'my_template.html'
    url_kwargs = [
        kwargs.String('my_string'),
        kwargs.Int('my_int', 42)
    ]
```

In this case, we have specified that the view takes two arguments, one string and one int. And there is a default value for the integer value.

The values of these arguments are provided as kwargs to the respective methods of the view as well as to the context.

From your template, here is how you would link to it:

```
{% load auto_url %}

<a href="{% url_from_class "my_app.views.MyView" my_string="Hello World" %}">Click me!
↪</a>
<a href="{% url_from_class "my_app.views.MyView" my_string="Hello" my_int=32 %}">
↪Click me, too!</a>
```

The respective python function provide a args and kwargs parameter you can use:

```
from my_app import views
from django_auto_url.urls import reverse_classname

# resolve now. views.MyView must have already been declared.
url_for_view = reverse_classname(views.MyView, kwargs = {
```

(continues on next page)

(continued from previous page)

```
                'my_string': 'Hello World'
            })
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index