The Django Framework in Python

Matt Waismann

August 13th, 2021

1 Introduction

Django can be installed via pip. There's a useful commands for starting a Django project:

- django-admin startproject *project_name*

This will create a directory with the project's name and within that four files:

1. __init__.py - an empty file
2. settings.py - this is where settings and configurations are set/changed
3. urls.py - this is where urls map users to different locations in the application
4. wsgi.py - this is how the python app and the web server communicate

In the same location as the project there will be another file:

1. manage.py - you will regularly run 'python manage.py runserver'. This sets up a version of your application on your local host. Tip: instead of using the IP address of your local machine to access your local host, you can use the alias 'localhost:¡port nubmer¿'

2 Applications and Routes

Django allows for multiple "apps" within a website. For example, there could be a blog portion of the website or a store part and they can each be their own app. This is great for including similar apps across multiple projects. To create a new app, run the following command in the directory that contains the manage.py file: *python manage.py startapp your_app_name*. The app creates a new directory with your app name and within that directory there should be several files. We will explore these later.

2.1 View.py and urls.py

We will see this file already imports render from django.shortcuts. We will also import

```
from django.HttpResponse import HttpResponse
```

Within this file we should create a function called 'home'. This will handle the traffic from the home page of the app. This is where the logic goes for how we want to handle certain routes. This function will return what we want the user to see when they are sent to this route.

```
def home(request):
    return HttpResponse('<h1>Blog Home</h1>')
```

We'll explain the 'request' parameter later. Since function will handle the traffic when users go to the home page, we have to map the url pattern to this view function. To do this we will create a 'urls.py' file inside the apps directory. This will be similar to the projects 'urls.py' file. It should look like this

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name = 'my\_app\_name\-home'),
]
```

We still need to go to our project's 'url.py' file and include the following lines

```
from django.urls import path, include

urlpatterns[
    path('admin/' admin.site.urls),
    path('my_app_name/', include('my_app_name.urls')),
]
```

This is necessary because we need to tell the whole web application which url maps to the app. From there, the *path* function will look inside the *my_app_name.urls* file to figure out where the rest of the url maps to. The *include* function simply tells path to exclude 'my_app_name/' from the string match it will attempt to do when looking inside 'my_app_name.urls' for the appropriate matching route.

Example

Suppose a user requests 'mydomainname.com/my_app_name/about/'. The web application will first look at the main/master 'urls.py' file in the project directory and check for 'my_app_name/'. Since the path is there, *path* then goes to the 'my_app_name.urls' file (with the truncated url) and then searches there for

the urlpattern 'about/'. If the urlpattern exists then the path function will look inside wherever that path function maps to. This could be another urls file for another app or a 'Views.py' function which returns something to the user. Our current app doesn't have a place to route 'about/' to so let's include something:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name = 'my\_app\_name\-home')
    path('about/', views.about, name = 'my\_app\_name\-about'),
]
```

This tells path to call the *views.about* function.

This may seem like unneccesary structure at first, but the benefit of using this heirarchical approach is that a developer only needs to change the parent 'node' (i.e. from my_app_name to my_app_name_2) and all of the urls which follow from there will automatically reroute. This makes the code more easily modifiable.

Note *(credit: https://stackoverflow.com/questions/42212122/why-django-urls-end-with-a-slash)*

The reason why a trailing '/' is used at the end of url pattern is most easily explained with an example:

Now the reason for a trailing slash, consider a view (in any framework) which relatively resolves about.html for a user at path, users/awesomeUser since users/awesomeUser and users/awesomeUser/ are different

If the user is at users/awesomeUser, the browser will resolve it as users/about.html because there ain't a trailing slash which we don't want
If the user is at users/awesomeUser/, the browser will resolve it as users/awesomeUser/about.html because there is a trailing slash