

Django Web Framework



Django Static Files

In the previous chapter we discussed about templates, template variable and tags.

Templates are used to define the structure of the web page. But to use CSS, image files we have the concept of static files. Just like template folder that contains our HTML files, we are going to create a static folder that will hold our CSS, JavaScript and images file.

The static folder will contain our files for styling and enhancing the user interface, the codes that will provide interactivity and dynamic behavior to your webpages and visual elements like images, logos and icons.

The reason why the static files are separate and kept in a different folder is-

- It maintains a clean separation of concerns, making your project easier to manage and navigate.
- Static files are generally cached by web browsers, which can significantly improve loading times for repeat visitors.

Django offers the **django.contrib.staticfiles** application to streamline static file management. It is already added in the `INSTALLED_APPS`, so we don't have to add it separately.

```
33  ▾ INSTALLED_APPS = [  
34      'django.contrib.admin',  
35      'django.contrib.auth',  
36      'django.contrib.contenttypes',  
37      'django.contrib.sessions',  
38      'django.contrib.messages',  
39      'django.contrib.staticfiles',  
40  ]
```

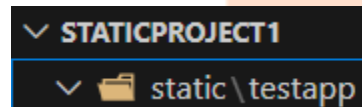
How to use and configure static files-

Step 1: Create a new project and application, configure app in settings.py-

```
C:\Users\LENOVO\Documents\djangocourse>django-admin startproject staticproject1
C:\Users\LENOVO\Documents\djangocourse>cd staticproject1
C:\Users\LENOVO\Documents\djangocourse\staticproject1>python manage.py startapp testapp
```

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

Step 2: Create a new folder named “**static**” inside the root directory.



```
▼ STATICPROJECT1
  ▼ static\testapp
```

Also, it's a good practice if you create the same name folder as application name inside the static folder to keep it separate in case of multiple applications.

Step 3: Now configure static files in settings.py-

```
import os
```

```
STATIC_URL = 'static/'
STATIC_DIR = os.path.join(BASE_DIR, 'static')

STATICFILES_DIRS=[
    STATIC_DIR,
]
```

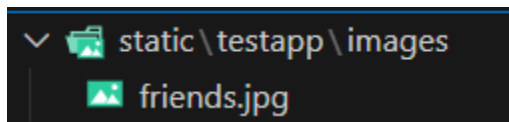
STATIC_DIR is a variable pointing to a custom directory where you place static files.

STATICFILES_DIRS is a Django setting that includes **STATIC_DIR** (among potentially other directories) to tell Django where to find additional static files.

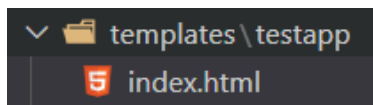
Step 4: To load an image on our template, create images folder inside static file.

static->testapp->images

Add a image of your choice inside the images folder-



Step 5: Create a template folder at the root directory and create a template named index.html. Configure template in settings.py. By using template tags we are going to load our static content on the web page.



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR, 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

Inside our html file, at the beginning of the file we have to include-

```
{%load static%}
```

This tag is essential for referencing static files (CSS, JavaScript, images, etc.) within your Django templates.

To reference the static files we use-

The {% static %} template tag within your Django templates is used to reference static files using their relative paths within the static folders

```

templates > testapp > index.html > html > body > img
1  <!DOCTYPE html>
2  {%load static%}
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>LOADING IMAGE</title>
8  </head>
9  <body>
10     
11 </body>
12 </html>

```

Step 6: Create a view to render our template and map it with a URL.

```

testapp > views.py
1  from django.shortcuts import render
2
3  # Create your views here.
4  def index_view(request):
5      return render(request, 'testapp/index.html')

```

```

testapp > urls.py
1  from django.urls import path
2  from testapp import views
3
4  urlpatterns = [
5      path('', views.index_view),
6  ]

```

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('testapp.urls'))
]

```

Step 7: Start the development server and visit the URL on a browser.

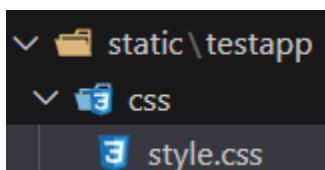
python manage.py runserver



LEARNING

How to load CSS Files?

Step 1: In the same project create a CSS folder in static->testapp and inside it create a CSS file named style.css.



Step 2: Add some styling in CSS file and update the template index.html-

To give reference to the css file we have to add-

```
<link rel="stylesheet" href="{%static 'testapp/css/style.css'%}">
```

If we want to reference to a CSS file we will use template tag `{%static%}`.

style.css-

```
static > testapp > css > style.css > img
1  img {
2      height: 500px;
3      width: 800px;
4  }
```

updated index.html-

```
templates > testapp > index.html > html > head > link
1  <!DOCTYPE html>
2  {%load static%}
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>LOADING IMAGE</title>
8      <link rel="stylesheet" href="{%static 'testapp/css/style.css'%}">
9  </head>
10 <body>
11     
12 </body>
13 </html>
```

Step 3: Start the development server and you can see the changes on the webpage.

On the server you can see-

```
[21/May/2024 22:43:04] "GET /static/testapp/css/style.css HTTP/1.1" 200 47
```

Here, if the response code was 404 that means that the server can't find the CSS file, and the reason can be some mistake in code, file URL or configuration.

In the next chapter, we will create a project that incorporates all the features we have learned so far, including templates, static files, views, and URLs.

If you are following the course series and learning the concepts easily, please subscribe to our YouTube channel. This will motivate us to create more educational, course videos and study material.

Join our growing community of tech enthusiasts! Subscribe to our YouTube channel, where we simplify programming languages in the easiest way possible. Gain a deeper understanding with our clear explanations and receive exclusive notes to enhance your learning journey. Don't miss out on valuable insights – hit that subscribe button now and embark on a programming adventure with us!

Subscribe to our channel:

<https://www.youtube.com/@HTEASYLEARNING>