Static Files (CSS, JS, mp4, jpg, ...)

To add static files like styles and scripts, we need to follow the same path as when we wanted to make templates. Inside the app directory we make a folder called `static` and inside it we add anotherfolder named the same as our app. The hierarchy for this project is shown below.

```
/MyProject
    /MyProject
    /user
        /templates/user
            index.html
        /static/user
            styles.css
            scripts.js
```

Make sure you have `django.contrib.staticfiles` added to project settings

```
INSTALLED_APPS = [
    'django.contrib.staticfiles',
]
```

The set up for this project is fairly simple. We have a Master Template file with basic HTML element, the `index.html` at user directory extends it and adds a "User Page" h1 title. the `styles.css` holds a background-color for the body element, the `scripts.js` holds a function called `hello()` which logs "Hello, World!" in console. Now let's see how we can render the styles and scripts alongside html!

First off, take a look at project settings, you'll find something like this:

```
STATIC_URL = 'static/'
```

Now let's take a closer look to the Master Template & HTML file we want to render with styles:

```
<!-- master.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    {% block head_ref %}{% endblock head_ref %}
    <title>{% block page_title %}Document{% endblock page_title
%}</title>
</head>
<body>
    {% block body %} {% endblock body %}
```

```
</body>
</html>

<!-- index.html -->
{% extends "master.html" %}
{% load static %} <!-- Pay attention to load -->

{% block head_ref %}
    <link rel="stylesheet" href="{% static 'user/styles.css' %}">
{% endblock head_ref %}


{% block page_title %}
User
{% endblock page_title %}

{% block body %}
<h1>
    User Page
</h1>
{% endblock body %}
```

And that'll do the work.

One not so tiny detail here is that the number of HTML and CSS files increase massively as the project grows. It is very important how they are foldered and named. Basically the entire project hierarchy is one crutial thing to keep in mind. Below you'll find an AI generated documentation of how a standard Django Project Hierarchy must look like.

Best Practices for Naming and Organizing Files

1. **Use Consistent and Descriptive Names**
   - Name files clearly based on their purpose or the feature they support.
   - Avoid overly generic names like `styles.css` or `script.js`.
2. **Keep Files Organized by App**
   - Group files by app to maintain modularity.
   - Place static files (CSS/JS/images) in a `static` directory within each app.
   - Place templates (HTML files) in a `templates` directory within each app.
3. **Namespace Static Files**
   - Use subfolders or prefixes to avoid filename collisions (e.g., `css/base.css` or `js/admin.js`).
4. **Leverage the Django Template and Static Structure**
   - For templates, keep files under the `templates/<app_name>/` directory.
   - For static files, keep them under `static/<app_name>/`.

Example Project Structure

Here’s a scalable Django project directory structure:

Let’s work through a real-world example of a Django project. Let’s imagine we’re building a **Learning Management System (LMS)**, with features like managing courses, user profiles, and discussions.

```
lms_project/
├── manage.py
├── lms_project/                     # Project settings directory
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py                 # Project settings
│   ├── urls.py                     # Root URL configuration
│   ├── wsgi.py
├── requirements.txt                # Dependencies
├── static/                         # Global/static files (shared
assets)
│   ├── css/
│   │   ├── global.css              # Global styles for the entire
project
│   ├── js/
│   │   ├── global.js               # Shared JavaScript logic
│   ├── images/
│   │   ├── favicon.ico             # Shared favicon
├── templates/                      # Shared/global templates
│   ├── master.html                   # Global base template for the
whole project
│   ├── 404.html                    # Custom 404 page
│   ├── navbar.html                 # Shared navbar template
├── courses/                        # App for managing courses
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   │   ├── __init__.py
│   ├── models.py                   # Models for courses
│   ├── static/                     # Static files for the courses app
│   │   ├── courses/                # Namespace for app-specific
static files
│   │   │   ├── css/
│   │   │   │   ├── courses.css     # Styles for the courses section
│   │   │   ├── js/
│   │   │   │   ├── courses.js      # JavaScript for course
interactions
│   │   │   ├── images/
│   │   │   │   ├── course_banner.jpg
│   ├── templates/                  # Templates for the courses app
│   │   ├── courses/                # Namespace for templates
│   │   │   ├── base.html         # Base template for course pages
│   │   │   ├── course_list.html    # Page listing all courses
│   │   │   ├── course_detail.html  # Individual course page
│   ├── tests.py
│   ├── urls.py                 # URLs specific to courses
│   ├── views.py                # Views for course functionality
├── users/                      # App for managing user profiles
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   │   ├── __init__.py
│   ├── models.py               # Models for user profiles
│   ├── static/                 # Static files for the users app
│   │   ├── users/              # Namespace for app-specific
static files
│   │   │   ├── css/
│   │   │   │   ├── profile.css     # Styles for user profile pages
│   │   │   ├── js/
│   │   │   │   ├── profile.js      # JavaScript for user profile
actions
│   │   │   ├── images/
│   │   │   │   ├── avatar_default.png
│   ├── templates/              # Templates for the users app
│   │   ├── users/              # Namespace for templates
│   │   │   ├── login.html      # User login page
│   │   │   ├── signup.html     # User signup page
│   │   │   ├── profile.html    # User profile page
│   ├── tests.py
│   ├── urls.py                 # URLs specific to users
│   ├── views.py                # Views for user functionality
├── discussions/                # App for managing discussions
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   │   ├── __init__.py
│   ├── models.py               # Models for discussion threads
and comments
│   ├── static/                 # Static files for the discussions
app
│   │   ├── discussions/        # Namespace for app-specific
static files
│   │   │   ├── css/
│   │   │   │   ├── discussions.css # Styles for discussion pages
│   │   │   ├── js/
│   │   │   │   ├── discussions.js  # JavaScript for discussion
interactions
│   │   │   ├── images/
│   ├── │   │   ├── thread_icon.png
│   ├── templates/                  # Templates for the discussions
app
│   │   ├── discussions/            # Namespace for templates
│   │   │   ├── thread_list.html    # List of discussion threads
│   │   │   ├── thread_detail.html  # Detail page for a single thread
│   ├── tests.py
│   ├── urls.py                     # URLs specific to discussions
│   ├── views.py                    # Views for discussion
functionality
```

Key Features of this Structure

1. **Realistic App Names**
   - Each app corresponds to a real feature of the LMS (e.g., `courses`, `users`, `discussions`).
2. **App-Specific Organization**
   - Each app has its own `static` and `templates` directories, and files are namespaced to avoid conflicts. For example:
     - `static/courses/css/courses.css`
     - `templates/courses/course_list.html`
3. **Global vs. App-Specific Assets**
   - **Global assets** like `global.css`, `global.js`, and `base.html` are stored in top-level `static/` and `templates/` directories.
   - **App-specific assets** are kept within the app's `static/` and `templates/` directories.
4. **URL Modularization**
   - Each app has its own `urls.py` file, which is included in the main `urls.py` using Django's `include()` function. For example:

```python
from django.urls import path, include

urlpatterns = [
    path('courses/', include('courses.urls')),
    path('users/', include('users.urls')),
    path('discussions/', include('discussions.urls')),
]
```

5. **Ease of Scaling**
   - If a new feature, such as notifications, needs to be added, you can simply create a `notifications` app with its own static files and templates.

Thanks to ChatGPT for providing that knowledge, note that a lot of the things you see in here might be too advanced just for now. They'll come clear soon enough.

Master Static Files

Just like the master template, you can have a master style for your project. We did this in the current project by making a folder at the project root `/MyProject/static`. in which we'll store master styles.

In that folder we added a `master.css` file in which we made the page font-family to `Courier New`. Now we must add the mentioned css file to the `master.html` file we made earlier.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet" href="{% static 'master.css' %}">
    {% block head_ref %}{% endblock head_ref %}
    <title>{% block page_title %}Document{% endblock page_title
%}</title>
</head>
<body>
    {% block body %} {% endblock body %}
</body>
</html>
```

One last step is to add the global static files directory to the project settings like this:

```
from MyProject.MyProject.settings import BASE_DIR

STATIC_URL = 'static/'
STATICFILES_DIRS = [
    BASE_DIR / 'static'
]
```