Member-only story

FastHTML with Tailwind CSS (no

Learn how to use the Tailwind CSS CLI to generate custom CSS files for your FastHTML app without Node.js.

Coding Pit · Following

8 min read · Aug 20, 2024

192 2

As a full-stack framework, FastHTML is designed to keep JavaScript and CSS out of the picture, leaving the developer with just pure Python code. However, there will be times when the frontend needs of a project will outgrow the included Pico CSS framework or any other minimalistic CSS framework. At that point, the full-stack FastHTML Python developer will need to work with a more sophisticated CSS framework, and Tailwind CSS might just be the best one available.

What is Tailwind CSS?

Tailwind CSS is a utility-first CSS framework. Instead of writing custom CSS for each component, you apply predefined classes directly to your HTML elements. Each class is designed to style a specific aspect, like margins, padding, colors, fonts, etc.

Here's how it differs from traditional CSS:

- **No custom stylesheets**: Instead of writing CSS rules in external files, you use Tailwind's predefined utility classes in your HTML.
- **Rapid development**: You style elements by composing small utility classes, allowing quick design iterations without diving into custom CSS.
- **Customization**: Tailwind is highly customizable via configuration. You can override default styles and add your own values.
- **Responsive design**: It has built-in support for responsive design, with utilities like `sm:` , `md:` , etc., to apply different styles based on screen size.

Example Button:

```
<button class="bg-blue-500 text-white py-2 px-4 rounded">
Click Me
</button>
```

In this case:

- `bg-blue-500` sets the background color
- `text-white` sets the text color
- `py-2 px-4` applies padding
- `rounded` adds rounded corners

What's the Issue?

Just like with the Pico CSS framework, you could include Tailwind CSS using a CDN. The URL for that is https://cdn.tailwindcss.com.
However, on Tailwind's official site, it states:

> *Use the Play CDN to try Tailwind right in the browser without any build step. The Play CDN is designed for development purposes only and is not the best choice for production.*
> *Source.*

One of the great advantages of Tailwind CSS is that it generates the CSS file on the fly based on the actual use of the utility classes in the app. This means it leaves out CSS code for utility classes that aren't even used in your app, making the resulting CSS file very slim and completely tailored and optimized to your app.

Sounds good? It really is. So let’s get to it…

Get Tailwind CSS Standalone CLI

In typical web development, you would use Node.js and its packaging system npm to get Tailwind CSS up and running. It seems a bit excessive to install and run npm for just a single package, though. Luckily, Tailwind CSS comes with a standalone CLI for every platform, so no Node.js is needed.

Go to the release section on Tailwind’s GitHub page, find the latest release version (usually the first one listed), and pick the right installer for your platform. Install it on your machine. Make sure you can easily access the CLI from anywhere in the terminal. Linux users might want to add an alias to the `.bashrc`, while Windows users should ensure it’s in the PATH environment variable.

Try it by simply running `tailwindcss`, and the result should look something like this:

```
$ tailwindcss

tailwindcss v3.4.10

Usage:
   tailwindcss [--input input.css] [--output output.css] [--watch] [options...]
   tailwindcss init [--full] [--postcss] [options...]

Commands:
    init [options]

  ....
```

Setup FastHTML

First, make sure Python 3.10+ is up and running. Create a `fasthtml-tailwind` directory and `cd` into it.

```
mkdir fasthtml-tailwind
cd fasthtml-tailwind
```

Create a Python virtual environment and activate it. For this example, I will use good old `venv` .

```
python -m venv .venv
source .venv/bin/activate
```

Now install `python-fasthtml` using pip:

```
pip install python-fasthtml
```

FastHTML is ready. Let’s test it.
In the IDE of your choice, create a `main.py` file in the root folder `fasthtml-tailwind` that we created earlier. Add the simple FastHTML app setup to `main.py`:

```
# main.py

from fasthtml.common import *

app, rt = fast_app()

@rt('/')
def get():
    return Div('Hello World!')

serve()
```

Now, let’s run the run the dev server:

```
python main.py
```

Go to `http://localhost:5001` in your browser, and you should see "Hello World!" on your screen.
For those familiar with FastHTML, nothing exciting so far. So let’s move on with Tailwind CSS…

Setup Tailwind CSS

Go back to the terminal, into your root folder, and create two new folders called `src` and `public`.

```
mkdir src public
```

We'll use the `src` folder, as the name suggests, as the source for Tailwind CSS to work from. The `public` folder will be where the CLI will drop the generated CSS file, which we will then link to in our app.

Now back to the terminal. In our root folder, use `tailwindcss init`, which should result in a success message saying a config file has been created.

```
tailwindcss init

Created Tailwind CSS config file: tailwind.config.js
```

Open that `tailwind.config.js` file in your IDE, and it will look like this:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

The important part for now is the `content` field. That's where we tell Tailwind which files have the utility classes that Tailwind should generate its CSS code from. In our case, it's just the .py files in our root directory. So let's add them and save the config file:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["*.py"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Be aware that the `content` field needs to be adjusted for different file and folder structures. For example, if your `.py` files are in subfolders within a `pages` folder, add `"./pages/**/*.py"`.

Now, we need to give Tailwind an input CSS file. So, create an `app.css` file in the `src` folder and add the following code:

```
/* app.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

That's it. This seems a bit tedious, as this input CSS never actually differs, but that's the process.

Now let's start the watcher. This will essentially go through all Python files, look for Tailwind CSS utility classes, and generate the final `app.css` file in our `public` folder, which we can then use in our app—all within a blink of an eye.

Run the following line from the root folder in terminal. Since this will be continously watching, make sure to keep the terminal alive.

```
tailwindcss -i ./src/app.css -o ./public/app.css --watch
```

You will get a warning saying:

```
warn - No utility classes were detected in your source file ...
```

No worries. We haven't used any utility classes in our app yet, nor have we actually included the generated `app.css` file yet. But simply having the watcher running, we should already have an `app.css` file in our `public` folder. Great, let's connect the last pieces…

Bringing it all together

First, let's include the generated `app.css` file in our FastHTML app. Open up your `main.py` file again and change the `app` variable:

```
app = FastHTML(hdrs=Link(rel="stylesheet", href="app.css", type="text/css"))
rt = app.route
```

Using the `FastHTML` class rather than the `fast_app` function, removes Pico CSS, so we don't have to disable it. We add a stylesheet link to our `app.css` using the `hdrs` keyword. But wait… How does FastHTML know which `app.css` to pick? Well, it doesn't… so we have to tell it!

```
@rt("/{fname:path}.{ext:static}")
def get(fname:str, ext:str):
    return FileResponse(f'public/{fname}.{ext}')
```

This function takes any request for a file, adds the `public` folder to the filename, and returns it as a file. In our case, the link to the `app.css` file we added to the header, will now look for `/public/app.css`, which is exactly where it should be. A missing file will result in a 404 error, logged in the console. By the way, this folder structure and routing system is by no means set in stone. I personally prefer working with `/static` or `/asset` routes, but we'll keep it simple for now.

We can add some Tailwind CSS classes now. Let's say we want "Hello World!" to be in large red text. Go to the `main.py` file and change:

```
@rt('/')
def get():
    return Div('Hello World!')
```

to:

```
@rt('/')
def get():
    return Div('Hello World!', cls="text-3xl text-red-500")
```

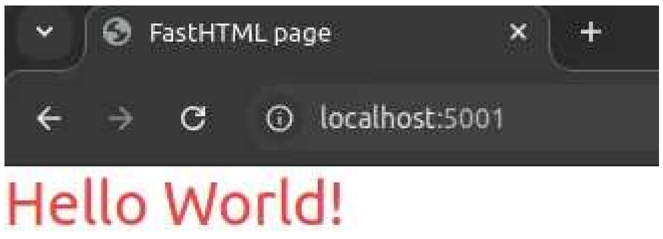
The complete `main.py` :

```
# main.py

from fasthtml.common import *

app = FastHTML(hdrs=Link(rel="stylesheet", href="app.css", type="text/css"))
rt = app.route

@rt("/{fname:path}.{ext:static}")
def get(fname:str, ext:str):
    return FileResponse(f'public/{fname}.{ext}')

@rt('/')
def get():
    return Div('Hello World!', cls="text-3xl text-red-500")


serve()
```

Run the dev server with .venv active!

```
python main.py
```

Check http://localhost:5001 !



That's it! We've successfully added Tailwind CSS into our dev stack! There's a bit more though…

Tailwind CSS Templates for FastHTML

Now that FastHTML and Tailwind CSS are working together, what's next? One of the biggest advantages of Tailwind CSS is the vast amount of **open-source** templates and components, ready to copy and paste into your projects.

Sites like Tailwindflex offer a wide range of templates. For example, visit a landing page template like this one: Landing Page Template. You can copy the HTML code from this template and integrate it into your FastHTML project.

Since the template will be in HTML, you'll need to convert it into FastHTML components. Doing this manually would be time-consuming, so FastHTML includes a function called `html2ft`. This utility converts HTML templates into FastHTML components directly. I've written a small python CLI, that let's you do the conversion in the command line. You can download it from here or copy the following lines and save it as html2ft.py in your root folder

html2ft.py code from Github Gist

And here's how to use it:

Copy the HTML content into a `page.html` file and place it in your `src` folder.

Run the `html2ft.py` by executing:

```
python html2ft.py page
```

This will generate a `page.py` file containing the FastHTML components equivalent of your HTML, wrapped in a variable called `content` .

Once generated, you can import this `page.py` file into your `main.py` and render the content:

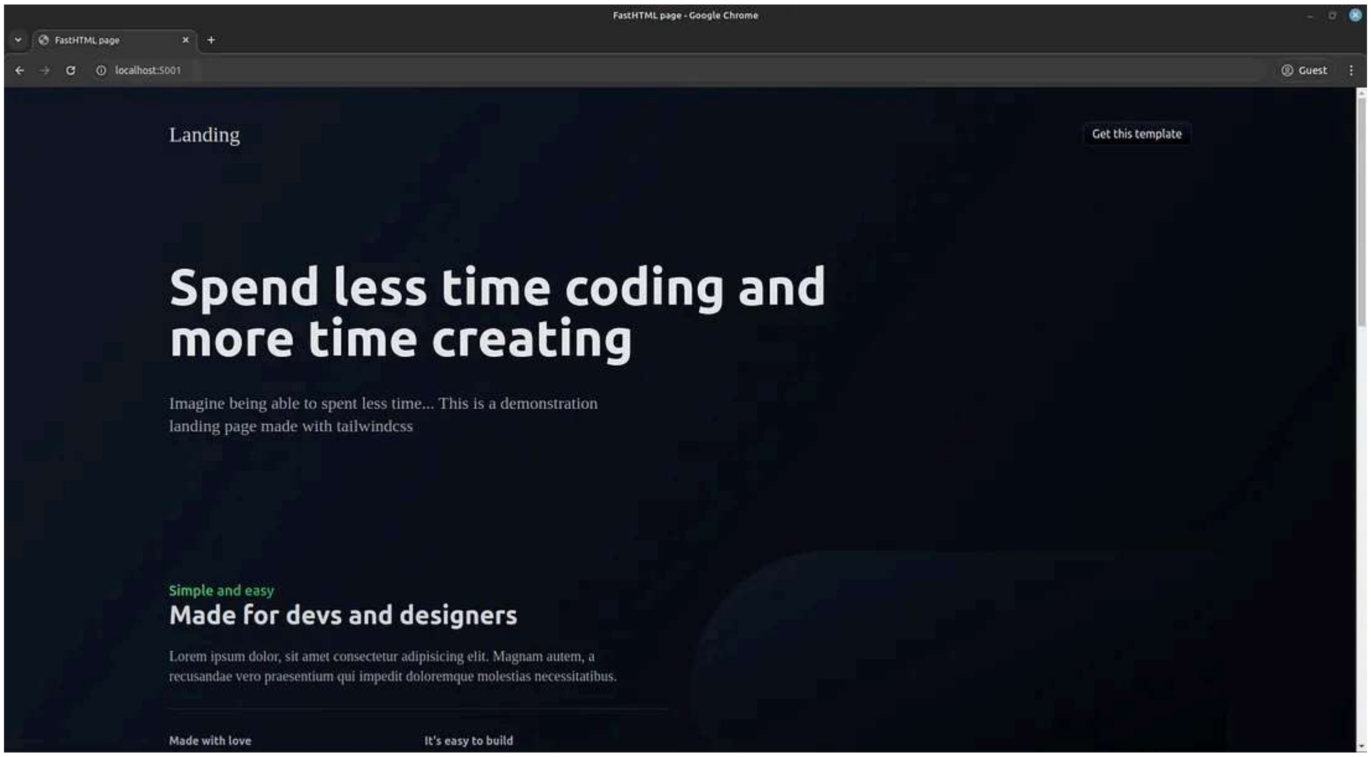
```python
# main.py
import page

...

@rt('/')
def get():
    return page.content
```

The complete `main.py` :

```python
# main.py

from fasthtml.common import *
import page

app = FastHTML(hdrs=Link(rel="stylesheet", href="app.css", type="text/css"))
```

```
rt = app.route

@rt("/{fname:path}.{ext:static}")
def get(fname:str, ext:str):
    return FileResponse(f'public/{fname}.{ext}')

@rt('/')
def get():
    return page.content

serve()
```

And finally, check if the Tailwind CSS CLI watcher is still running, since we've added quite a few things for him to convert. Run the dev server again with .venv active and check the result on http://localhost:5001/



Spend less time coding and more time creating

Imagine being able to spent less time... This is a demonstration landing page made with tailwindcss

Simple and easy

Made for devs and designers

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magnam autem, a recusandae vero praesentium qui impedit doloremque molestias necessitatibus.

There you go! Beautiful FastHTML app build with Tailwind CSS!

Deployment Tips

At some point, your project will need to move from development to production. Here's a quick tip. Tailwind comes with a built-in optimizer to remove unnecessary CSS classes from the generated file and compress it to make it as small as possible.

Simply run:

```
tailwindcss -i ./src/app.css -o ./public/app.min.css --minify
```

Then, change the linked CSS file in your `main.py` file from `app.css` to `app.min.css`, and you are ready to deploy.

Also make sure you add the `src` folder to the `.gitignore` and ***exclude*** it from any deployment and production step. It will be filled with templates and assets that are no longer needed.

Conclusion

It's great that FastHTML comes with the Pico CSS framework, allowing you to get started without having to think too much about the frontend. For small projects, this will suffice. However, FastHTML is also suitable for larger projects, where Tailwind CSS will likely become a go-to option due to its flexibility and sophistication.

This guide demonstrated how to integrate Tailwind CSS into the FastHTML development stack without needing Node.js, by using the standalone Tailwind CSS CLI. Additionally, it showed how to use FastHTML's `html2ft` function to convert Tailwind HTML templates into FastHTML components, enabling a streamlined workflow.

With this setup, you can leverage Tailwind's powerful utility classes while working within the FastHTML environment, ensuring your app's frontend is both optimized and maintainable.

Python Fasthtml Tailwind Css Front End Development No Javascript

Coding Pit

Written by Coding Pit

24 Followers · 29 Following

Following

I'm a Technical Director in the VFX industry with a strong passion for coding and a keen interest in web development.

Responses (2)