You Can Serve Static Data Over HTTP

Published on 2022-01-27 20:00:00+01:00

This will be short. I need to let some steam off and flush this thing out of my mind because it's already hanging around for too long. The story goes like this: we're prototyping small tool for whatever and we're at the point where there is a visible need to store some (most likely) static data. Oh, the thing is in JavaScript by the way.

"Yeah, let's just put it into a JSON file and serve it over web server."

Said me naively but turns out that there is a legitimate confusion on the other side of the trench.

"Do you mean a database?"

Let's end the flashback here, we already have enough background for what will follow. Now, hear me out, *your API endpoint is just pretending to be a file.*

You don't need a full blown web application back-end. You don't need an over-featured database engine that can provide its own REST API endpoints with data. And you don't need a complex ORM library and couple your data model to it and write your own endpoints. You don't need any of these and possibly others.

You *may* need them, but never let your habits decide your designs.

It's not illegal to use static files and it does work. Moreover, with a limited set of data it gives huge freedom to developer to try out things and break them. It's suited for rapid prototyping but it also has its own place in production environment (e.g., git).

Is it possible to learn this power?

Not from a modern developer. You need to jump down the rabbit hole of the history of solid and proven solutions.

First off, create some files. In my example we needed some static data for JavaScript, so we used JSON:

```
[
        {"type": 1, "name": "a"},
```

```
        {"type": 1, "name": "b"}
]
```

Now, save it as a file called `things` without any extension at all. Something like this will do:

```
$ pwd
/home/ignore/public

$ ls
things
```

It's ready to be served to the client. Start up your favourite server that can handle static data and position it at the directory that you've selected:

```
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

It can now be accessed at `http://localhost:8000/things`. If you are working with JS as I did that time, shift the `public` content a bit to also serve the client application:

```
$ ls . api
.:
index.html  style.css  main.js  api/

api:
things
```

Dead simple, right? Now you know. Hey, you can even mock more complex APIs with a help of symlinks and creativity. And if you worry about whether it works - just try it out. If it has some problems - force the server to set correct mime type for files without extensions.

Hopefully, next post will be more interesting, but this had to be done. Just had to. I may rest now.