6. Building a web application

Disclaimer:

This post will get into the guts of how I've built a prototype of the application I've been talking about for the last several posts. If you're not interested in hearing about how Python, Flask, and Vue.js work, it might be best to skim this and move on to the next post as quickly as possible.

The Guts:

I mentioned earlier that this application was going to be a client-server app. What does that look like in practice? Well, it starts about as simply as possible, with two folders named `client` and `server` (and one more called `venv` which I encourage you to read about here).

```
root@Liam-XPS:webapp # ls -l
total 0
drwxrwxrwx 1 root root 512 Dec 12 15:35 client
drwxrwxrwx 1 root root 512 Dec 13 10:36 server
drwxrwxrwx 1 root root 512 Dec 12 10:20 venv
```

What does this really mean, though? Under the hood, the client and the server are just processes running on some computer (ideally on separate computers, but since I only have one, I'll be running them both on my machine). So really, the client and server folders just contain code to run two processes which interact with each other.

Server File Structure

```
root@Liam-XPS:server # ls -l
total 12
-rwxrwxrwx 1 root root 2034 Dec 12 15:18 app.py
drwxrwxrwx 1 root root  512 Dec 13 10:56 data
drwxrwxrwx 1 root root  512 Dec 13 10:57 examples
drwxrwxrwx 1 root root  512 Dec 12 19:21 __pycache__
-rwxrwxrwx 1 root root    0 Dec 13 10:36 tweet_auth.py
-rwxrwxrwx 1 root root 1358 Dec 12 19:21 tweet_search.py
-rwxrwxrwx 1 root root  342 Dec 12 19:08 twitter_bot.py
-rwxrwxrwx 1 root root  311 Dec 12 10:19 voter_data_processor.py
-rwxrwxrwx 1 root root 1884 Dec 12 14:39 voter_data_search.py
```

As you can see, this is really a pretty simple architecture. There are some Python files (with suffix .py) which process voter data from the `data/` folder and some others which interact with the Twitter API. The crux of it all is the `app.py` file, a snippet of which I've included below. That file defines the *routes* of the server. That is, if the server can be found at `https://server.com`, the app.py file determines what happens when you navigate to `https://server.com/example_route`.

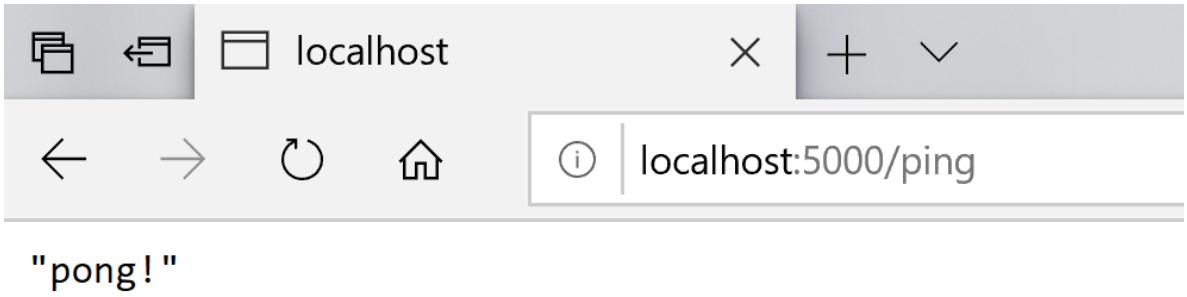
```
# instantiate the app
app = Flask(__name__)
app.config.from_object(__name__)

# sanity check route
@app.route('/ping', methods=['GET'])
def ping_pong():
    return jsonify('pong!')

# ... Obviously there's more to this than what I've included here

if __name__ == '__main__':
    app.run()
```

Of course, this isn't the full file and there's a lot more logic to it, but if you look at the section labeled "ping_pong," you'll see an example of the general structure of this server. When the user navigates to a certain route (like `server/ping`), the server displays some data (in this case, just the word "pong!"). The data is returned in JSON format, which is just a simple and easy-to-parse format which you can read about here.



Like so!

Client File Structure

```
(venv) root@Liam-XPS:src # ls -l
total 4
-rwxrwxrwx 1 root root 114 Dec 12 15:35 App.vue
drwxrwxrwx 1 root root 512 Dec 12 10:19 assets
drwxrwxrwx 1 root root 512 Dec 12 10:19 components
-rwxrwxrwx 1 root root 341 Dec 12 15:37 main.js
-rwxrwxrwx 1 root root 593 Dec 12 11:39 router.js
drwxrwxrwx 1 root root 512 Dec 12 10:19 views
```

This is a little bit harder to understand than the Python, but it's the same general idea. The `main.js` file creates the `App.vue`, which is the main visual piece of the client application, as well as the `router.js`, which fulfills the same route-creation function I described above. The `components` and `views` pieces define the smaller sections of the complete application (like menus, toolbars, and layout files), while the `assets` folder contains necessary data for the application (like sound files or images). The client then simply makes requests to the server for the data it needs to display. I am indebted to this tutorial by Michael Herman for helping me understand the Vue side of this, and I recommend it to anyone looking to understand this more fully.

Why Both?

It may seem like overkill to run two applications with very similar structures, but as I mentioned earlier, there are compelling security and engineering reasons to use this kind of paradigm. The most powerful reason, though, is simply that different programming languages and frameworks have different strengths. Vue.js is a fantastic framework for visual displays in the browser, while Python is far more versatile for querying APIs and processing data. Using both allows each to focus on what it excels at. Enough of this, though – let's see it in action!

App Demo

https://youtu.be/2LfmW7Y2L_I