Flask (along with Werkzeug and socket)

General Information & Licensing

Code Repository	https://github.com/pallets/flask https://github.com/pallets/werkzeug
License Type	BSD 3-Clause "New" or "Revised" License
License Description	● A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the copyright holder or its contributors to promote derived products without written consent. ● Essentially, this license just means you're allowed to use the current repository but you aren't allowed to warrant it off as your own and promoting the creator and any contributors to individuals without getting consent from these individuals.
License Restrictions	● Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. ● Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. ● Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Magic ⋆★˚₊° ⋆.° ☽ °⌒☄｡°★彡⋆ ༄

Flask starts our server off with a line like:

app.run(host="0.0.0.0", port=5000, debug=True)

This line will start our connection to a TCP server and create a TCP connection for us. To start off, we begin with creating the Flask server/environment which sets up our local development server. This specifies that our app is going to be a Flask application. All of this is happening in app.py in the flask code between from line 1129 all the way to 1193.

However, the lines from 1129 - 1184 are really there to just set up the flask application. The important thing happens in line 1188 where we use a function import from Werkzeug in order to run our application. Therefore, the next thing that gets called is *run_simple* from the Werkzeug code.

From here, we go into the *run_simple* function which essentially (based on the comments) will help start a development server for the WSGI application. This function runs from lines 1048 - 1099 in the serving.py file from the Werkzeug repository.

A lot of the steps from 1051 - 1069 in the serving.py file seem to be more like sanity checks to make sure our port is an actual number and kind of sanitizing what needs to be done.

The big chunk is the *make_server* function that gets assigned to srv which makes me feel like this is definitely important and a place where they (obviously) make servers. When going into the function, it specifies that the return value is going to be of type BaseWSGIServer. So, now I know that a server is getting returned which means this is definitely the right place.

Going through the function, we enter the *if threaded* conditional statement since we do have a threaded server and this returns a new instance of *ThreadedWSGIServer* BUT since we need to also pass in a *BaseWSGIServer*, we create a new *BaseWSGIServer*. *BaseWSGIServer* takes in a HTTPServer Object (after importing it from the http library and server.py file), which links back to the HTTPServer Class in the cpython server.py file in the HTTP library (lines 130-140). As best as I can tell, the *HTTPServer* class is just a wrapper around Python's lower level TCP Socket Server libraries. Which means that when HTTPServer takes in the TCP socket it behaves in a manner similar to what we learned in the course, and this can be seen here, where it imports the *socketserver* library. Now, the *HTTPServer* class uses this socket and binds it to itself creating an object reference to the socket. It then configures its server name and port. And then when we backtrace, the *BaseWSGIServer* class does the same configuration with the HTTPServer class as the HTTPServer class does with *socketserver*. Basically, long story short it's all a matter of object references progressively adding more and more functionality as we go up until *BaseWSGIServer*, which represents one of the fundamental concepts of computer programming- abstraction.

The next biggest thing that happens and probably the most important is line 702 - 705 in serving.py. This line takes us to the socketserver library and creates a TCP instance in lines 445 - 513. At this point, once *BaseServer* is initialized, we have successfully created the TCP connection since this TCP class binds the socket and then activates it. At which point, we will return back to the call from *make*_server and we now have a server that has not been started up yet!

The next couple of stack frames then take us through activating and starting up the server and printing out the link which can be accessed and providing any warnings the develops of Flask would like you to know when running your development server in addition to setting up a logger which I don't feel like was talked about in this course I will no go into detail about that.

Finally, we check the reloader to make sure we don't need to reload the server in any way. If not, we just call the *server_forever()* function that will serve the server we just created and that is how we create a TCP connection on a socket and start a server.

Link To Specific Code (in the order in which they were accessed)

(assuming we start from `app.run(host="0.0.0.0", port=5000, debug=True)`)

1. https://github.com/pallets/flask/blob/main/src/flask/app.py (the line numbers are a bit messed up from my package and Github but the line number here is 1067 for the *run()* function)
2. https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py (the line numbers 907 - 1069 for *run_simple()*)
3. https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py (the line numbers from 853 - 895 for *make_server()*)
4. https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py (the line number 651 - 812 for *BaseWSGIServer* and line number 815 - 823 for *ThreadedWSGIServer*
5. From socketserver library in python, we call the TCPServer to create a new TCPServer