Open-Source Report: TCP Connections

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

Flask

General Information & Licensing

Code Repository	Flask Repository
License Type	BSD 3-Clause "New" or "Revised" License via source
License Description	● All code is free for commercial and private use ● Code is free to be modified ● Code is free to distribute ● No citations or credits are required when publishing code that uses code from this repo
License Restrictions	● Flask is not liable if we use their code and it breaks anything ● Flask does not provide a warranty for their code

Magic ⋆★˚｡⋆˚☽˚⁀➷｡˚★彡⋆༄

Our project uses Flask to establish the TCP Connection. Our code, called ‘socketio.run’ with our Flask app as a parameter. ‘socketio.run’ calls Flask’s run function in flask_socketio’s ‘__init__.py’ file. Flask has a function called “run” that runs our server on a given port and host IP-address. It does this by taking a host IP and a port as parameters (there's more but not as relevant) and then uses those inputs to run a function called “run_simple” from the BaseWSGIServer (which extends HTTPServer, which inherits TCPServer from socketserver.py) class in serving.py from the werkzeug library.

This function is what actually calls the ‘serve_forever’ function on the created server. To make the server object, ‘run_simple’ calls a function “make_server” that takes the params passed into ‘run’ and returns an instantiation of a BaseWSGIServer. The BaseWSGIServer object then initializes itself with the ‘__init__’ function. This function actually calls the super function ‘__init__’ of the ‘TCPServer’ class from ‘socketserver.py’ which also calls the ‘__init__’ function of the ‘BaseServer’ class from the same library to set the server_address.

Once initialized, the ‘TCPServer’ init function binds the host and port to the server object with the functions ‘server_bind’ defined in the class. The ‘server_bind’ function sets the socket’s ‘server_name’ to the actual usable hostname based on the given IP-address and then sets the attribute ‘server_port’ to the given port number. After that, the ‘__init__’ function runs ‘server_activate’. The function ‘server_activate’ calls function ‘listen’ from the ‘socket.py’ file to listen for data on the specified port and host which creates the server as we know it. The serve_forever function polls for server shutdown requests over the given poll interval. When a server shutdown request is received, the server is terminated.

socketio’s ‘run’ function

Our Flask function in our code starts from here:

Here we are trying to initialize our SocketIO instance where we put our Flask app in our parameters and we’re setting namespace to “item” and cors_allowed_origins to origins

Flask 'socketio.run' function: The starting point of the Flask web application with SocketIO, this function initializes the application, sets the debug mode, and allows unsafe Werkzeug. It is called in SocketIO's ‘__init__.py’ file. Where Flask’s ‘run’ function is called in socketio’s ‘__init__.py’:

Flask ‘run’ function: This function is responsible for starting the Flask web application, which in turn calls the `run_simple` function from the Werkzeug library.

Where Flask calls ‘run_simple’: The `run` function then calls the `run_simple` function from the Werkzeug library.

Where werkzeug defines ‘run_simple’: Part of the Werkzeug library, a core component of Flask, this function sets up the web server and handles incoming requests.

Where make_server is defined: This function is called within `run_simple` and creates an instance of the `BaseWSGIServer` class, which is a subclass of Werkzeug's `HTTPServer` class.

Where BaseWSGIServer is defined: This class defines the WSGI server used by Flask and inherits from the `HTTPServer` class, which is part of the Python standard library.

Where HTTPServer is defined in server.py (from python Libs): Defined in `server.py`, this class is part of the Python standard library and provides the foundation for the WSGI server.

Where TCPServer is defined in socketserver.py: Defined in `socketserver.py`, this class provides TCP socket handling functionality and is a superclass of the `HTTPServer` class.

Where init is defined in BaseServer class: This method initializes the `BaseServer` class, setting up the server address and request handler.

Where server_bind and server_activate are called in TCPServer: These methods are called within the `TCPServer` class to bind the server to the provided address and activate it to start accepting connections.

Where server_bind() is defined in TCPServer class
Where server_activate is defined in TCPServer class
Where werkzeug defines super function ‘serve_forever:
Where werkzeug calls ‘serve_forever’ in ‘run_simple’

Where serve_forever is defined in socketserver.py: This method is called within `run_simple` and is responsible for starting the server loop that listens for incoming connections and processes requests.

The method calls interact in the following manner:

- The `socketio.run` function initializes the Flask application and calls the Flask `run` function.
- The Flask `run` function calls Werkzeug's `run_simple` function.
- `run_simple` sets up the server and calls `make_server`, which creates an instance of `BaseWSGIServer`.
- `BaseWSGIServer` inherits from `HTTPServer`, which in turn inherits from `TCPServer`.
- The `TCPServer` class calls the `server_bind` and `server_activate` methods to set up the server.
- Finally, the `serve_forever` method in Werkzeug starts the server loop, processing incoming requests.

The following shows a method call hierarchy with class locations and inheritance:

```
Flask.run (Flask class, Flask library)
└── Werkzeug.run_simple (werkzeug.serving module)
    └── Werkzeug.make_server (werkzeug.serving module)
        └── BaseWSGIServer.__init__ (BaseWSGIServer class, werkzeug.serving module)
            └── HTTPServer.__init__ (HTTPServer class, http.server module)
                └── socketserver.TCPServer.__init__ (TCPServer class, socketserver module)
                    └── BaseWSGIServer.serve_forever (BaseWSGIServer class, werkzeug.serving module)
                        └── socketserver.serve_forever (TCPServer class, socketserver module)
```