Open-Source Report

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	https://github.com/pallets/flask
License Type	BSD 3-Clause
License Description	• Permissive, can use without restriction • Redistributions must meet a few extra conditions • The copyright holder is not liable for anything
License Restrictions	• Cannot use name of copyright holders or names of contributors for endorsing or promoting our product without written permission • Redistribution requires listing a copyright notice and list of conditions •

Magic ★★。˚⋆˚☽˚｡⋆˚★彡☆༄

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section will likely grow beyond the page

- TCP connections
    - Must include the code, in your chosen framework from the list above, where the connections are established or where a library approved on the homework is called

Werkzeug:
https://github.com/pallets/werkzeug/

Flask:
https://github.com/pallets/flask

TCP connections

1. First, the Flask.run() function is called on the Flask object that was created

   Our code that calls app.run():
   https://github.com/enochand/CSE312Project/blob/main/main.py#L108

   Run() function Inside of flask library:
   https://github.com/pallets/flask/blob/main/src/flask/app.py#L786

2. The run function from above then calls run_simple() from werkzeug.serving:
   https://github.com/pallets/flask/blob/main/src/flask/app.py#L902

   The code for the run_simple() function can be seen here:
   https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L938

3. The run_simple function creates a BaseWSGIServer server using the make_server

function in the same file. For our simple application this base server is a ThreadedWSGIServer which inherits from the BaseWSGIServer.

Run_simple calling make_server function:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L1068

Make_server function inside werkzeug:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L884

Returning ThreadedWSGIServer:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L908

4. A BaseWSGIServer inherits from the HTTPServer class that is located in the http library server.py file. This HTTPServer is initialized by the super()__init__() function call in the initialization of the BaseWSGIServer class..

   BaseWSGIServer class:
   https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L682

   Function call of super()__init__() that creates HTTPServer class:
   https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L732

   HTTPServer class that BaseWSGIServer inherits from:
   https://github.com/python/cpython/blob/main/Lib/http/server.py#L130

5. When the HTTPSserver is created it inherits from the TCPServer located in the socketserver.py file.

   TCPServer from socketserver.py that HTTPServer inherits from:
   https://github.com/python/cpython/blob/main/Lib/socketserver.py#L390

   TCPServer initializing base server:
   https://github.com/python/cpython/blob/main/Lib/socketserver.py#L450

   BaseServer from socketserver.py that TCPserver inherits from:
   https://github.com/python/cpython/blob/main/Lib/socketserver.py#L153

6. This BaseServer is the low level server that receives requests and passes them up the chain. This server has useful functions like server_forever, which calls the _handle_request_noblock function whenever there is a new request. _handle_request_noblock gets the request from the TCP connection with the get request get_request() function. Get_request function calls self.socket.recvfrom() to get the request.

   Server_forever:

https://github.com/python/cpython/blob/main/Lib/socketserver.py#L216

Server_forever calling _handle_reqeust_noblock:
https://github.com/python/cpython/blob/main/Lib/socketserver.py#L238

Get_request call inside of _handle_request_noblock:
https://github.com/python/cpython/blob/main/Lib/socketserver.py#L311

```
    def get_request(self):
        data, client_addr = self.socket.recvfrom(self.max_packet_size)
        return (data, self.socket), client_addr
```

```
    def recv(self, __bufsize: int, __flags: int = ...) -> bytes: ...
    def recvfrom(self, __bufsize: int, __flags: int = ...) -> tuple[bytes, _RetAddress]: ...
```

This is deeper than we needed to go, but Jesse said we wouldn't lose points for going too deep. When all of this is built at the end of the run_simple function, run_simple calls the svr.serve_forever() function on the ThreadedWSGIServer it created. This makes the server loop it's always checking for new requests. When it gets a request it runs _handle_request_noblock as discussed in step 6.

Run_simple calling serve_forever() function:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L1100

```
        )
    else:
        srv.serve_forever()
```