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	pallets/flask: The Python micro framework for building web applications. (github.com)
License Type	BSD 3-clause "New" or "Revised" License
License Description	• Allows for commercial use, modification, distribution and private use • Removes liability from the owner • Copyright is owned by Pallets
License Restrictions	• Liability • Warranty

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚｡⋆ ★ ⋆

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

First is flask/app.py line from 773 to 894 (flask/app.py at main · pallets/flask · GitHub) which is basically a bunch of check to make sure the input arguments are correct, checks to see if its run using flasks own command line tool and then runs run_simple from the werkzeug library

Then in the werkzeug library file serving.py (werkzeug/serving.py at main · pallets/werkzeug · GitHub(this is referred to many times)) lines 945 to 1107 which then sees if it needs static files or not and does other checks, until it runs make server on line number 1075 which jumps to line 891 in the same file which creates a either threaded or non threaded wsgi server, for the non threaded server is line 689 of serving.py (for the full object declaration) which on init creates an httpserver from the python standard library server.py line 130 (cpython/server.py at 3.11 · python/cpython · GitHub) and inherits the socketserver.TCPServer which is used in the homework.
The original basewsgi server has the function serve_forever called on it which just calls the httpserver serve_forever (since it inherits the httpserver it uses super()) which just calls the tcpserver's serve_forever function directly (yet again inheritance) which is how the tcp server is created(the threaded and fork version of basewsgi just use the socketserver fork or threaded variants)

The http request is handled at line 389 of file serving.py in WSGIRequestHandler class of werkzeug, which calls its super().handle() or tries to handle any errors which calls BaseHTTPRequestHandler handle function which is on line 428 of pythons standard library (import http) server.py which calls handle_one_request on line 391 of the same file which checks to make sure the request isn't too large, and is not None or len 0, otherwise it checks to make sure it was properly parsed through a call to parse_request on line 267 of the same file. Which checks the http version or send a http.BadRequest error, then parses the headers which is done through the function http.client.parse_headers on line 224 of file client.py (cpython/client.py at 3.11 · python/cpython · GitHub) which then calls _read_headers on line

206 which basically uses file io to read each line into a list and returns the list which is then joined and passed into email.parser.Parser on line 16 which creates a feedparser (cpython/feedparser.py at 3.11 · python/cpython · GitHub), sets it to headers only and feeds which splits the lines at line 101 which is called from the feed function at 173 all the data into it and closes it which parses the last of the data which create an enumerate object on the lines and returns them in a dict after parsing the headers by finding the ‘:’ in each header splitting it and adding it in.