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.

[insert library/framework name here]

General Information & Licensing

Code Repository	https://github.com/Ai-Jesse/Blue-Jesse
License Type	MIT
License Description	● Any user can obtain this software free of charge ● Any user is allowed to copy, modify, and distribute ● Users are allowed to sell the software
License Restrictions	● The software is provided "as is" without warranty ● The authors or copyright holders can't be liable for any claim, damages, or other liability

Magic

Parsing HTTP Headers using Flask:

We first create our server using **app = Flask(name)** in our code. Flask is the library that we are using, which includes parsing the HTTP Headers.

```
# Setting up the App
app = Flask(name)
```

From the Flask library, we import **request**, which is part of the globals.py module of Flask.

```
from flask import Flask, render_template, redirect, request, url_for, session, make_response
```

request is of type **Request** from Flask wrappers.py module. This **Request** class also extends the **Request** class from werkzeug library's wrapper/request.py module. Which also extends the **Request** class from werkzeug's sansio/request.py module.

Flask wrapper Request:
Werkzeug wrapper request:
Werkzeug sansio Request:

The **Request** class from sansio/request.py module have the **self.headers** attribute which stores all of the HTTP headers after its parsed.

The step to parse the requests are as follows:

After establishing TCP connection and receives a request from the client, the Flask server uses the socketserver library's **ThreadingMixIn** class to process requests asynchronously.

ThreadingMixIn: https://github.com/python/cpython/blob/main/Lib/socketserver.py#L671

The method **finish_request** from ThreadingMixIn is called which takes in the request. Then it creates a **WSGIRequestHandler** from werkzeug servering.py module. **WSGIRequestHandler** extends **BaseHTTPRequestHandler** class from http library's server.py module, which extends the **StreamRequestHandler** from socketserver library, which then extends **BaseRequestHandler** from socketserver.

finish_request: https://github.com/python/cpython/blob/main/Lib/socketserver.py#L358

WSGIRequestHandler:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L148

BaseHTTPRequestHandler:
https://github.com/python/cpython/blob/main/Lib/http/server.py#L146

StreamRequestHandler:
https://github.com/python/cpython/blob/main/Lib/socketserver.py#L776

BaseRequestHandler:
https://github.com/python/cpython/blob/main/Lib/socketserver.py#L730

When initializing the **WSGIRequestHandler**, it calls the **handle** method, which then calls the **handle_one_request** method, which then calls the **parse_request** method from **BaseHTTPRequestHandler**, as part of this **parse_request** method, it calls the

parse_headers method from http library's client.py module, and set the headers to **self.headers** attribute

handle: https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L358

handle_one_request: https://github.com/python/cpython/blob/main/Lib/http/server.py#L391

parse_request: https://github.com/python/cpython/blob/main/Lib/http/server.py#L267

parse_headers: https://github.com/python/cpython/blob/main/Lib/http/client.py#L224

The **handle_one_request** method then calls the **run_wsgi** method from werkzeug's serving.py module. It then calls the **make_environ** method, which creates a **WSGIEnvironement** variable called **environ** that contains the header and other information.

run_wsgi: https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L237

make_environ:
https://github.com/pallets/werkzeug/blob/main/src/werkzeug/serving.py#L159

Afterward **run_wsgi** calls the execute method which calls the **__call__** method of the Flask object, which then calls the **wsgi_app** method.

__call__: https://github.com/pallets/flask/blob/main/src/flask/app.py#L2546

wsgi_app: https://github.com/pallets/flask/blob/main/src/flask/app.py#L2498

According to Flask documentation, **wsgi_app** method is called to handle each request, it creates a **RequestContext** using the **request_context** method. This **RequestContext** object then have a **self.request** attribute which is a **Request** object that we mentioned in the very beginning. This **request** attribute then stores all of the information from the **environ** variable that was passed around.

RequestContext: https://github.com/pallets/flask/blob/main/src/flask/ctx.py#L278

request_context: https://github.com/pallets/flask/blob/main/src/flask/app.py#L2426

This then allows us to use the request object in our server, and we can access the headers using **request.headers**.

In short, when the Flask server receives a request, it creates some handler object to handle and parse it, then creates the request object that we can use.