CSE 312 Project Open Source Reports

NAMES: Paige Shoemaker, Brett Sitzman, Vincent Lin, Micheal Hoste, Ian Hunter

You are required to write 3 open-source reports as part of your project. These reports will be on the following features of your framework:

- 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
- Parsing HTTP headers
    - Must include the code in your framework from the above list that parses the headers
- WebSockets
    - Be sure to include how the connection is established as well as how frames are parsed in the library you chose for WebSocket connections

For each report, you must trace through the libraries you use until you find the code that actually does the work that you want it to do (eg. You must provide links to the code in the library that resembles your HW code). You will show this entire trace of calls from your code to the code doing the work which may include calls through many classes and libraries. Be prepared to dig deep into these libraries!

Uvicorn TCP connections

General Information & Licensing

Code Repository	https://github.com/encode/uvicorn
License Type	BSD 3-Clause
License Description	• Similar to BSD 2-Clause License, but with a 3rd clause that prohibits use of the copyright owner or any contributors to promote products derived from the library without written consent
License Restrictions	• Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the 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

How it Works
Uvicorn uses the “loop.create_server()” function from python’s built-in asyncio library to create TCP servers. This function uses python’s built in socket library to create a socket and perform the 3-way handshake.

Chain of Calls
The chain of calls begins in line 290 of server.py in our project repository, where the TCP server is started using the “uvicorn.run()” function

Uvicorn’s run() function begins on line 460 of main.py, however the line we are most interested in is line 578, where they call the “server.run()” function

The run() function is defined on line 59 of server.py in the Uvicorn repository, and it calls the serve() function which is defined on line 63 of that file

The serve() function continues on to call the startup() function (defined on line 88), which is where the TCP connections are managed. These connections are created using the loop.create_server() function, which is a part of python’s built-in asyncio library

- The most important lines of the startup() function are 161-167, which is the standard scenario where new TCP connections will be established using a host/port pair
- Lines 138-140 are where existing sockets are reused to create new TCP connections
- Lines 129-131 are where TCP connections are setup when the server is run from a Gunicorn worker (Gunicorn is often used for load-balancing and stability when concurrency is being used, which can increase performance of the server)

The “loop.create_server()” function is defined on line 1444 of base_events.py python’s asyncio library. This function uses python’s built-in socket library to create TCP sockets. There are a few lines in particular that are important for the creation of a TCP server:

- On line 1513, the socket is created
- On line 1537, the socket is bound to an address
- On lines 1557-1561, the server is started using the established TCP connection

FastAPI Parsing HTTP headers

General Information & Licensing

Code Repository	https://github.com/tiangolo/fastapi
License Type	MIT License
License Description	• Anyone can obtain a free copy of FastAPI so long as it deals with software use. • They can use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software
License Restrictions	• No warranty and/or liability. • Copyright owners are not held responsible for any claim, damages, or other liabilities that may arise.

Magic

→ First, the TCP socket is created and the HTTP request is received from the client. Next, the HTTP request is parsed by the server into its components, such as the HTTP method, path, and headers. After the request is received and parsed, headers are stored in a dictionary-like object called ‘environ’. After the headers are stored, FastAPI uses the ‘Header’ class from the fastapi module and accesses the headers in the ‘environ’.

- HTTP request is sent out and received followed by an HTTP response within server.py on lines:
  - 29 - 34 https://github.com/MichaelHoste2020/CSE312-Project/blob/db9571e228f67006c164cf1aba85e342a146d8ea/server.py#LL30C30-L30C30
  - 37 - 42 https://github.com/MichaelHoste2020/CSE312-Project/blob/db9571e228f67006c164cf1aba85e342a146d8ea/server.py#L37
- Fast API uses the “Request” object from the starlette library to parse HTTP headers
  - The Request object is defined on line 190 of requests.py in the starlette repository
  - The headers are parsed on lines 110-114 in the same file

FastAPI WebSockets

General Information & Licensing

Code Repository	https://github.com/tiangolo/fastapi
License Type	MIT License
License Description	• Anyone can obtain a free copy of FastAPI so long as it deals with software use. • They can use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software
License Restrictions	• No warranty and/or liability. • Copyright owners are not held responsible for any claim, damages, or other liabilities that may arise.

Magic

→ Its purpose is to create a websocket connection to the server. A programmer would create a new fastapi instance by calling app = fastapi().

- From here, use app.websocket() to establish a path for the websocket. When this is called it will add a new websocket route, the function is located at https://github.com/tiangolo/fastapi/blob/d59c27d017a805fcf847aa624b5edd7e4659bbe0/fastapi/applications.py#L402.
- The WebSocket is defined by starlette and acts as a HTTPConnection https://github.com/encode/starlette/blob/24c1fac62a80bb153c6548145334fc643991e35a/starlette/websockets.py#L21
- The HTTPConnection contains http data defined in the following class https://github.com/encode/starlette/blob/24c1fac62a80bb153c6548145334fc643991e35a/starlette/requests.py#L63.
- Now that we have a route stored in the server we can reference it whenever we want to send data. Inside starlette to send a text message over websockets you would use the .send_text function located at https://github.com/encode/starlette/blob/24c1fac62a80bb153c6548145334fc643991e35a/starlette/websockets.py#L162 on the websocket. This will send a formatted string with all of your chosen data.