Open-Source Report (Websockets)

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_sock

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-sock
License Type	MIT License
License Description	• Redistributions are permitted with or without modification • Including Commercial uses • Any redistribution must maintain the same license
License Restrictions	• Contributors are not liable for any damages caused by the software • Names of contributors shall not be used to promote derived software w/o permission

Magic

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
    - WebSocket Handshake: After the TCP connection is established, the front-end sends a HTTP request to upgrade the connection to a WebSocket connection. This request contains a hashed key, “Sec-Websocket-Key”, whose unhashed form is a random string of characters. The server then appends the string "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" to the key, hashes the new string and encodes it into base 64. The server then sends the updated string under the header, “Sec-Websocket-Accept”. From now onward, the server and client will be sending Websocket frames over TCP instead of HTML requests.
    - WebSocket Packets: Of the requests sent, all but those with opcode “0xA” (signaling a closing connection) send byte data between the server and client, and the data necessary to parse it. Namely, a xor-mask (if MASK=1), payload size, opcode, and FIN (which determines whether the current frame is the end of a string of subsequent frames containing related data). These packets allow the server to interact with the client and allow both to dynamically update each others’ state.

- 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.
    - Initialization: As flask_sock is a Flask plugin, it takes a Flask app as an argument
        - “Sock.__init__” method called at NN-CSE312/app.py: line 24 with flask app as parameter
        - Sock.__init__, (lines 12-19) in our case, just takes the Flask app provided and stores a reference to it as a member “self.app”
    - Socket Route & handshake
        - HTTP Request is received from the client
        - flask_sock/Sock decorator method “route” used at NN-CSE312/app.py: line 175
        - Decorator in question
        - flask_sock line 56: Server is instantiated and borrows the TCP socket here
        - line 73: Server method “handshake” called simple_websocket/ws.py: lines 326-334
        - WSConnection “receive_data” -> H11Handshake “receive_data” -> H11 Connection “recieve_data” carrying request
        - wsproto/handshake.py line 170: “_process_connection_request” gets headers and other related data. Turns it into a wsproto/Request object
        - _handle_events (simple_websocket/ws.py 172-249) is then called (see Recieving)
        - line 179: wsproto/H11Handshake method “send” wsproto/handshake.py: lines 91-114
        - line 105: wsproto/H11Handshake method “_accept” in wsproto/handshake.py: lines 254-291
        - line 260: wsproto Function “generate_accept_token” in wsproto/utilities.py: lines 85-88
        - **Macro for GUID** in wsproto/utilities.py: ACCEPT_GUID in line 18

        - In, _accept, sends constructed request, via h11, here
    - Receiving
        - simple-websocket/Base method (as simple-websocket/Server) (see above) “receive” is called at NN-CSE312/app.py: line 181
        - simple_websocket/ws.py lines 99-115
        - line 109: receive busy-waits until it receives something in input_buffer
        - In simple-websocket/Base _handle_events, any events received from the client end up in the input_buffer here (ws.py line 230)
        - _handle_events iterates over the Generator received from WSConnections “events” which yields to wsproto/Connection "events"
        - In WSConnection’s _proto, (FrameProtocol) __init__ calls method “_parse_more-gen” which calls method “process_buffer” which calls “parse_header”
        - **Parsing code for websocket frame**: wsproto/frame_protocol.py 395-446 called here
        - In a separate thread: **the socket's recv is called** and is then passed through the chain of "recieve_data/bytes" to Add a number of bytes to a buffer
    - Sending
        - simple-websocket/Base (as simple-websocket/Server) method “send” is called at NN-CSE312/app.py: line 195 and 202
        - Sends a Message Class to ws.send:
        - A Lovely Call Stack
        - Until the Frame Protocol prepares to serialize the data (checking what kind of data it’s sending) wsproto/frame_protocol.py 593-614
        - **Serializing frame code (with an alarming number of helper functions)**: wsproto/frame_protocol.py 623-673
        - **Sending data to TCP socket**: simple_websocket/ws.py 97 (See TCP Connection Report)
    - Closing
        - simple-websocket/Base (as simple-websocket/Server) method “close” is called at NN-CSE312/app.py: line 189
        - Similar to send, but…
        - simple_websocket/ws.py 125: Sends a CloseConnection class to ws.send instead of Message
        - **Sending data to TCP socket**: simple_websocket/ws.py 128