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-sock

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-sock
License Type	MIT License
License Description	• A short and simple permissive license with conditions only requiring preservation of copyright and license notices. • Permission is granted, free of charge, for any person to obtain a copy of the software, and to deal in the Software without restriction. This includes without limitation the rights to copy, modify, merge, publish, distribute, sublicense, and sell copies of the Software, following the license restrictions listed below.
License Restrictions	• The same copyright notice and permission notice is included in all copies or substantial portions of the Software. • Software is provided, 'as is' without any warranty of any kind. Authors of copyright holders are not liable for any claim, damages, or other liability, arising from from or out of connection with the software or the use of other

	dealings in the software.

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
• 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

What it needs to cover

- 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

Useful Notes: The flask-sock packages make use of some of the event classes inside the Python310.site-packages.wsproto.events.py file. Of these I listed a few that are used and important to our implementation.

1. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/events.py#L224 This is the TextMessage class event that gets fired when the payload is a TEXT payload (op code 0x1).
2. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/events.py#L242 This is the BytesMessage class event that gets fired when the payload is a BINARY payload (op code 0x2).
3. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/events.py#L24 This Request class event is fired when the server receives a WS handshake request. It is normally followed by the creation of a AcceptConnection event.
4. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/events.py#L65 This AcceptConnection class event is used by the server when it wants to accept an upgrade request.

How Connection is established:

When a sock decorator is created, the websocket_route function creates an instance of a Server which is located in the simple_websocket.ws.py file.

1. Creation of server in websocket_rote function: https://github.com/miguelgrinberg/flask-sock/blob/main/src/flask_sock/__init__.py#L59
2. This is the Server class that is created. It inherits from the Base class which is inside the same file: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L253
3. Base class in same file that Server inherits from: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L39

The Base server creates the persistent WebSocket connection and stores it in the variable self.ws. This is done by creating a WSConnection class that is located inside the wsproto library.

4. Creation of WSConnection inside the Base Server: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L70
5. WSConnection class that is created: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L17

This WSConnection class completes the WebSocket handshake by using the H11Handshake class located in the wsproto.handshake.py file. This H11Handshake class is stored in the self.hanshake variable.

6. Creation of H11HandShake class in WSConnection class: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L30
7. H11HandShake class that is created: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L39

In the WSConnection class, the connection variable is initialized to None. The first time WSConnection.send() is called (i.e the self.connection variable is still None and the event is AcceptConnection), the H11Handshake.send function is called with event == AcceptConnection event.

8. WSConnection.connection initialized to None: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L31
9. First call of send, when WSConnection.connection is still == None: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L60
10. Call to WSConnection.handshake.send, which is the same as H11Handshake.send function. The event passed into the this function is an AcceptConneciton event: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L61
11. The H11Handshake.send() function that is called: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L91

When the event passed into H11Handshake.send() == AcceptConnection, H11Handshake calls the ._accept function.

12. Call to H11Handshake._accept():
    https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L105

The H11Hanshake class generates the response for the websocket upgrade request with the H11Hanshake._accept function. This _accept function takes in an AcceptConnection event as input, generates all the headers needed for the response, and sends the response to the client. The 3 important steps to this are, extracting the Sec-WebSocket-Key, generating Sec-WebSocket-Accept value, and creating all the headers for the response. Links to each of these steps are provided below.

13. Extracting the Sec-WebSocket-Key:
    https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L259
14. Generating Sec-WebSocket-Accept value:
    https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L260
    a. To generate the Sec-WebSocket-Accept value, the generate_accept_token function is used and can be seen here:
       https://github.com/python-hyper/wsproto/blob/main/src/wsproto/utilities.py#L85
15. Creating all the headers for the response:
    https://github.com/python-hyper/wsproto/blob/main/src/wsproto/handshake.py#L262

When this response is returned, both the server and client know they should keep a persistent WebSocket Connection between each other.

How Frames are Parsed

The frames are read and parsed in the wsproto.frame_protocol.py file:
https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py

After the Base Server (from step 3 of How Connection Is Established above) completes the websocket handshake, it creates a thread, and then calls thread.start() when thread.start() is called, the function passed into the thread at creation is run, so in this case the function self._thread is run. This causes the server's thread to continuously check for new data on the TCP connection.

1. Thread Creation, function self._thread is passed into thread class at creation:
   https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L75
2. Call to thread.start():
   https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L76
3. self._thread() function that the thread runs:
   https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L139

This function continuously loops while the base server connection is active. In this loop it constantly checks for new data at the TCP connection with the self.sock.recv() function.

4. https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L160

The data returned from this is the raw WS Frame bits and it needs to be parsed, unmasked, and could potentially contain a fraction of a frame or more than one frame. To unmaks and parse the frame, the WSConnection.receive_data function is called with the raw data received passed in as input.

5. Call to WSConnection.recieve_data: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L168
6. Function that is called in wsproto.__init__.py file: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L67

Because this is WSConnection.connection variable was set (to a Connection Class) when completing the WebSocket Handshake, the the input data is passed into the WSConnection.connection.receive_data function.

7. WSConnection.connection.recieve_data call: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L80
8. Connection class stored at WSConnection.connection: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L59
9. Connection.recieve_data function that is called with step 7: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L112

The Connection class has an attribute called called Connection._proto which is a FrameProtocol object. This FrameProtocol object also has a function called FrameProtocol.receive_bytes() that accepts the bytes of a WebSocket Frame and parses it.

10. Call to FrameProtocol.receive_bytes function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L133
11. FrameProtocol class: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L494
12. FrameProtocol.receive_bytes function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L560

The FrameProtocol class is initialized, which has an attribute FrameProtocol._frame_decoder which is an instance of the FrameDecoder() class. This class has a function called receive_bytes also, which stores the incoming data in a buffer to be parsed later.

13. Call to FrameProtocol.FrameDecoder.recieve_bytes function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L561
14. FrameDecoder class: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L327
15. FrameDecoder.reciveve_bytes function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L342
16. Buffer class. This class holds the data and has some useful functions like consume_exactly and consume_at_most. These functions are used when bit parsing to read the next desired amount of bytes of data conveniently. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L253

The stack frames are then popped and returned all the way back to the Base._thread() function that is running. After the data is added to the buffer, the Base._handle_events() function is called. This is what begins the process of parsing the frames stored in the buffer.

17. Base._handle_events() call: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L169
18. Definition of Base._handle_events() function: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L173

The Base._handle_events then iterates over all the events stored in the generator returned by the WSConnection.events() function.

19. Call to WSConnection.events() function: https://github.com/miguelgrinberg/simple-websocket/blob/main/src/simple_websocket/ws.py#L176
20. Function that is called in WSConnection class: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L82

This function calls the Connection.events() function stored in the WSConnection.connection attribute.

21. Callto Connection.events() function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/__init__.py#L91
22. Definition of Connection.events() function that is called: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L139

Inside the Connection.events function, the FrameProtocol.received_frames() function is called. This function parses and returns all of the frames currently on the buffer (where the data was finally stored on step 16).

23. Call to FrameProtocol.received_frames() function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L150

24. FrameProtocol.receive_frames() function definition: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L563

The FrameProtocol.receive_frames() function gets each frame from a call to the FrameProtocol._parse_more function. It weirdly defines the FrameProtocol._parse_more function to be equal to the FrameProtocol._parse_more_gen() function when the FrameProtocol is initialized, not sure why it has 2 names for the same function call.

25. Call to FrameProtocol._parse_more(): https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L564
26. Where FrameProtocol._parse_more is set equal to the FrameProtocol._parse_more_gen() function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L502
27. FrameProtocol._parse_more_gen() function definition: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L539

The FrameProtocol._parse_more_gen() has a while loop for decoding all the frames on the buffer. Inside the while loop each frame on the buffer is processed one by one by making a call to the FrameDecoder.process_buffer() function call. FrameDecoder.process_buffer() reads all the data for a frame off the buffer, and returns it in a convenient Frame() class.

28. Important while loop that is used to combine all the payloads of multiple frames, if the data sent happens to span multiple frames: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L548
29. Call to FrameDecoder.process_buffer() function: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L549
30. FrameDecoder.process_buffer() function definition. https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L345
    a. This function does all of the heavy lifting when it comes to processing one frame. This function figures out the fin bit, opcode, payload length, and mask with a call to the FrameDecoder.parse_header() call: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L347
    b. FrameDecoder.parse_header function definition: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L395
    c. Consuming 2 bytes off the buffer to get fin bit, reserve bits, opcode, and first piece encoding the payload length (either a value < 126, 126, or 127). https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L396

    d. Parsing fin bit: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L401
    e. Parsing reserve bits: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L402
    f. Parsing opcode bits: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L407
    g. Parsing mask bit, 1 if mask exits: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L416
    h. Parsing first piece of payload length: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L417
    i. Getting the rest of the payload length if the payload length input is 126 or 127: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L418
    j. Getting the masking key if the masking bit is 1: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L430
31. After all of this is figured out, the payload can be extracted. This is done here: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L358
32. If the payload is masked, it is unmasked with this call here: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L366
33. If the data for this message spans across multiple frames, then the payload from all the next frames including the fin frame must be combined. This is done here:
    a. Getting payload of next frame: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L377
    b. Combining payload of multiple frames: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L381
34. After all of this, the frame data is conveniently placed in a Frame class and returned.
    a. Placing data of frame in Frame Clas: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L384
    b. Frame Class Definition: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L230
    c. Returning frame class created: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L393

A little more upkeep code is run, for example decoding the frame from utf-8 bytes to a string if it is a text frame, but the final Frame class is processed and ready to be returned. This frame is added to the generator object returned from the FrameProtocol._parse_more_gen() function, and the next frame is processed if it exits on the buffer.

35. Adding frame to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L558

Popping off many function calls off the stack back to the Connection.events() call, each frame in the generator returned is given an event type. Either Ping, Pong, CloseConnection, TextMessage, or BytesMessage. These events types are final form our library uses, thus the frames are fully parsed.

36. Adding Ping event to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L154
37. Adding Pong event to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L159
38. Adding CloseConnection event to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L168
39. Adding TextMessage event to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L172
40. Adding BytesMessage event to generator returned: https://github.com/python-hyper/wsproto/blob/main/src/wsproto/connection.py#L180