flask-sock (along with simple_websocket and wsproto)

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-sock https://github.com/miguelgrinberg/simple-websocket
License Type	MIT License
License Description	● Commercial use ● Modification ● Distribution ● Private use ● A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.
License Restrictions	● Liability ● Warranty ●

Magic ⋆★｡˚⋆｡˚☽˚｡⋆˚★⋆｡˚

Websocket Handshake:

The process first starts off when we finally receive a GET request that asks us to upgrade our server into a websocket.

From here, using the flask-sock library, we have a route named sock.route(<oath>) that will handle our websocket duties since sock is our app that is able to utilize websockets. The websocket gets passed through as data to this route and the code (which is found in flask-sock under the decorator function creates a new instance of Server and this is where the handshake happens.

We move into the ws.py file which is part of the simple_websocket directory into a class called *Server* which takes our WSGI server or base server as a parameter and starts the process of first figuring out if our server was made from Werkzeug (which it was) and then proceeds to call the super class init function with a bunch of variables.

From here, we start initializing the variables needed to create the server and define if we are connected or not. All this code is not so important for us until we get to the line where we assign *ws* to a new *WSConnection*. This moves into another new directory by the name of wsproto and goes into our *init* function of the *WSConnection* class, where we assign our handshake to be a new instance of *H11Handshake*, which will basically just set our state for where our connection currently stands.

Once this is finished, we go back to the ws.py file and right under our most important line, we have our second most important part where we call the handshake method on our new websocket server!

Next, we go into a handshake method since we made our server a new instance of a websocket server and we go through this handshake method. As we go through, we noticed that we're just taking the headers and kind of splitting them and getting them ready to send them to a *receive_data* method which gets called in line 333 of ws.py.

We then move back into our *__init__.py* of the wsproto directory into our *WSConnection* class. Here, we call *receive_data* on our handshake object, which will put our headers into this object now (details not relevant).

Then, we get put into this *while True* loop where we keep calling the next event until we either get a connection was closed, the connection got paused or we need data.

(I would love to get into this part…really I would, but also this part is not a big part of what we're taught in this course so I'll give a brief summary here. Essentially, we are continuously going to process every event until we get a chance where our server is ready to send a response back AND we see that our client did actually send a request to open a connection and didn't close it right away.)

We then move into the *handshake.py* file where we start to process the connection request and split our headers up a lot more based on the key, value pairs from the tuples that were created. This process is basically just a for loop with a bunch of if statements and then making sure we aren't missing important headers that we need so I won't get into the details of this.

The important point is that we now made a new request with the headers we really wanted and that gets returned back to our *receive_data* method where we go through our while loop again. This second iteration just checks in case we need to switch protocols so this is not important as of right now and let's just fast forward to the 3rd iteration of this while loop.

At this point, we hit pause in our attempt to complete the connection event and now we because our pause is part of the if statement checking for our while loop, we can safely break out of the while loop.

At this point, we are still not connected since we never accepted the connection but since we processed it now, you can only guess what the next step is :)

We're back in the *ws.py* file and now we are changing our connected variable! We call to handle all of our events now and there really should be only one event right now since we're only making the connection!

We go to _*handle_events* and since our event is a Request, we can go into the first if statement and send our websocket server an *AcceptConnection* object!

We go into the *send* method on line 48 and send our handshake object the data we have and then moves us into the *send* method in the *handshake.py* file on line 91 and because our event is an *AcceptConnection* event, we call the method _*accept* (aka, the juicy stuff)

At this point, what I am literally looking at is homework code of where we have a method that makes our accept token by using the GUID in a separate method called *generate_accept_token* and starts putting the headers that we want in our response. We also can go down to line 283 in the *handshake.py* file and see that we are sending a response code of 101 and sending that response. At this point, I think it's fair to stop in terms of getting the Websocket handshake. Time to find out how they parse frames.

Parsing Websocket Frames:

Websocket Handshake Links To Code:

- https://github.com/miguelgrinberg/flask-sock/blob/10e0b3bb05580b63433106ef1b5ab42a735846d6/src/flask_sock/__init__.py#L53
- https://github.com/miguelgrinberg/simple-websocket/blob/59b6694974d2c7cac07a764665950db00e58abbf/src/simple_websocket/ws.py#L252
- https://github.com/miguelgrinberg/simple-websocket/blob/59b6694974d2c7cac07a764665950db00e58abbf/src/simple_websocket/ws.py#L39

- https://github.com/python-hyper/wsproto/blob/a7d9ff0c34567b042d025db555595bb6234f89e8/src/wsproto/__init__.py#L17
- https://github.com/python-hyper/wsproto/blob/a7d9ff0c34567b042d025db555595bb6234f89e8/src/wsproto/handshake.py#L39
- https://github.com/python-hyper/wsproto/blob/a7d9ff0c34567b042d025db555595bb6234f89e8/src/wsproto/handshake.py#L183
- https://github.com/python-hyper/wsproto/blob/a7d9ff0c34567b042d025db555595bb6234f89e8/src/wsproto/handshake.py#L254
- https://github.com/python-hyper/wsproto/blob/a7d9ff0c34567b042d025db555595bb6234f89e8/src/wsproto/utilities.py#L85

Websocket Frame Parsing Links To Code:

-

- 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