WebSocket

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-sock
License Type	MIT
License Description	Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
License Restrictions	No restriction.

Flask-Sock

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➴｡˚★彡⋆

In this project, since we chose flask as the primary web frame, we implement the websocket which is compatible with flask. To avoid the long-polling circumstances happening, we decide to use flask-sock instead of flask-socketio.

In our server.py, we run **app = Flask(__name__, static_folder="./static/functions.js").** The flask constructor takes the name of the current module (__name__) as argument. The static_folder means it created a specific route that will serve functions.js.

sock = Sock(app).

- The Sock() function is imported from the flask_sock package and specifically from the Sock library. It directs to create a Sock class. https://github.com/miguelgrinberg/flask-sock/blob/10e0b3bb05580b63433106ef1b5ab42a735846d6/src/flask_sock/__init__.py#L6
- The Sock class instantiates the Flask-Sock extension and it takes a parameter *app*, which is the Flask application instance. If the parameter is not provided, it must be initialized later by calling the function Sock.init_app method.
- The Sock.init_app method mentioned above takes the instance of class (self) and the Flask instance (app) as arguments. If the Flask instance of the class is not initialized yet, it runs a register method to register the blueprint for the instance of the class. Blueprints instead provide separation at the Flask level, share application config and can change an application object as necessary with being registered. So it successfully registered a prime Flask instance for the server.
- Then we go back to the __init__ method inside the Sock class. The parameters of this method are the instance of class (self) and the Flask instance. In the method, the Flask instance has been set to None . Also, the blueprint of Flask instance is considered as None as well. If the flask instance is not empty, it will return to its original instance then it will be initialized again to guarantee it's a new instance. Otherwise, the blueprint of the class instance will be defined.
- The blueprint of the class instance will be defined to a class **Blueprint.** https://github.com/pallets/flask/blob/3dc6db9d0cfddcfb971c382b014bb56ac3761d3c/src/flask/blueprints.py#L121
- The Blueprint class represents a blueprint, a collection of routes and other app-related functions that can be registered on a real application later. The parameter of the **Blueprint** is **Scaffold,** which is also a class.
- The Scaffold class is a common behavior shared between class Flask and class Blueprints. The parameters are import_name, static_folder, static_url_path, template_folder and root_path.
- Firstly, it defines the static_folder to be a Union type of the class instance and none via **Optional** method and **Union** method. These two methods help define the union type for both static folder and static url path. The reason why to do that is a union has some special characteristics:
    - Unions of unions are flattened, e.g. Union[Union[int, str], float] == Union[int, str, float]
    - Unions of a single argument vanish, e.g. Union[int] == int # The constructor actually returns int
    - Redundant arguments are skipped, e.g.Union[int, str, int] == Union[int, str]

  - When comparing unions, the argument order is ignored, e.g.Union[int, str] == Union[str, int] https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/typing.py#L662

- Then a json_encoder is created as a union of Type of JSONEncoder and None. https://github.com/pallets/flask/blob/3dc6db9d0cfddcfb971c382b014bb56ac3761d3c/src/flask/scaffold.py#L82
- The JSONEncoder is a Extensible JSON encoder for Python data structures. It supports many types of objects in Python and JSON. https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L74
- The **init method** in JSONEncoder takes multiple boolean parameters to create output with correct type.
  - If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L110
  - If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L114
  - If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L118
  - If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L123
  - If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L128
  - If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L132
  - If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is ``None`` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L137
  - If specified, default is a function that gets called for objects that can't otherwise

be serialized. It should return a JSON encodable version of the object or raise a
``TypeError``.https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L142

- After initialization, the **default** method returns a serializable object for "o" or calls the base implementation in order to raise a "TypeError".https://github.com/python/cpython/blob/db115682bd639a2642c617f0b7d5b30cd7d7f472/Lib/json/encoder.py#L161
- Besides **default** and **init** methods, there are still many functions constructed inside the JSONEncoder class. Since we are not going to use all of them, we will not keep tracking all the sub methods. The JSONEncoder class serves flask.json.dumps specifically. https://github.com/pallets/flask/blob/3dc6db9d0cfddcfb971c382b014bb56ac3761d3c/src/flask/scaffold.py#L82
- Meanwhile, JSONDecoder serves flask.json.loads. It creates a class based on json, returning a union object to json_decoder. https://github.com/pallets/flask/blob/3dc6db9d0cfddcfb971c382b014bb56ac3761d3c/src/flask/scaffold.py#L89

In the Sock class, the init_app initializes the flask-socket extension. If the application instance was not passed, it will also register a blueprint for the application instance.

Now, in our server.py, after we initialized the sock as a Sock class of application instance, the prototype of websocket class is established. The sock instance has a route decorator, but unlike the HTTP request and response, the decorator provides the WebSocket protocol handshake which allows the route to speak WebSocket.

- The **decorator** method conducts a websocket_route method. Inside, ws (websocket) is configured as a Websocket Server class. Then it calls the wraps function from functools library to encapsulate the websocket server with arguments before closing the websocket server.
- It also defines a WebSocketResponse class in order to set up the connection while sending the response to websocket. A call function is implemented in the class. While the mode of websocket is equal to 'evenlet', it handles a empty list then return it. https://github.com/miguelgrinberg/flask-sock/blob/10e0b3bb05580b63433106ef1b5ab42a735846d6/src/flask_sock/__init__.py#L67

After that, the websocket object (ws) which is passed to the route represents the actual WebSocket connection. And a basic websocket is established. The server can exchange information using send() and receive() method with the client. In our monopoly game, we receive the data via websocket and name it *data*. Then we convert the *data* into a python dictionary in order to check if there is a message indicating the successful connection via key. If so, we append client data and send it to the active users. If the socket message doesn't exist, the server will start a new game with a new board for all the active users. Hence, the active players can be involved in the same game and get updated simultaneously.

Websocket Frame Parsing

The websocket frame needs to be parsed so that our server can send out the correct data to the client. Since the **Sock** library has done the part for us, we dug deeply into the libraries that **Sock** imported, figured out how the internal frame decoder parses the websocket frame. What's more, we would like to see how the buffer is handled to receive inputs. Inside the repository https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py, multiple classes are established to decode the websocket frame.

- The **Buffer** class is initialized as a bytearray(), if there are some initial bytes, they will be appended to the buffer. Then, the method consume_at_most and consume_exactly can extract the exact number of data bytes into data, put the data bytes as used bytes, so the data that is consumed already will not be grabbed from the buffer again even though it still exists inside the buffer.
- When the buffer is ready, it can be one of the parameters of the **FrameDecoder** class. The framedecoder can also receive the data and append it to the buffer of our server.https://github.com/python-hyper/wsproto/blob/main/src/wsproto/frame_protocol.py#L342
- If the buffer contains a websocket frame, the process_buffer method starts working on parsing the websocket frame. At first, it detects if there is a header exists in the *self* class, as well as detects the boolean of header parsing. If falses, it returns None and the buffer will not be processed. The header parsing method parses fin code, rsv codes and opcode by the boolean of data[0] and packed corresponding code. Then it checks if *mask* exists and store the payload length, the method returns false if payload doesn't exist. If *mask* exists, takes the next 4 bytes in the buffer which are masking key and extract the masking key into self,marker. After the procedure is done, the header of the frame which has been decoded will be committed by the buffer. The buffer cleans out the parsed part and the method returns true.
- Now we are back to the part of the processing buffer. If the header is successfully parsed, the process will compare the length of buffer and the length of required payload. If the length of required payload is greater than the length of buffer, it means the buffer contains all required data and it has to keep reading the data. Otherwise, the payload will be extracted from buffer and processed with the four masking keys which are stored in self.marker via *process* method. The *process* method Xors one data byte with one masking key in one turn to decode the data, like what we've achieved in HW3. After finishing the decoding task, the real payload is translated to the message we want. The new websocket frame will be wrapped as a named tuple via **Frame** class instead of a raw websocket. The new frame contains the elements of opcode, decoded payload, payload length and fin code). The server can grab the payload or other information directly from the tuple frame easily.

Hence, the websocket frame is parsed successfully. The buffer keeps receiving new websocket frame, parses it in the way mentioned above and cleans out the data from the buffer after parsing is done. With the buffer and frame decoder, the server easily sends the correct data to multiple clients through the websocket connection and our game can run as expected!