

[Flask SocketIO Websocket]

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py
License Type	MIT License
License Description	<ul style="list-style-type: none">• Permits redistribution and use with/without modification• Permits distribution under different sublicense• Does not require distribution of source code
License Restrictions	<ul style="list-style-type: none">• Redistribution must preserve copyright and license notice• No forms of liability or warranty responsibility is implied by the author

Magic ★★°°☾°°👉°°★☸️🌟

```
✓ CALL STACK
  _handle_event      server.py
  _handle_eio_message serv...
  _trigger_event     server.py
  receive            socket.py 59:1
  _websocket_handler socke...
  __call__           gevent.py 44:1
  _upgrade_websocket socke...
  handle_get_request socke...
  handle_request     server.py
  handle_request     server.py
  __call__           middleware.py
  __call__           __init__.py 43:1
  __call__           app.py 2213:1
  run_websocket      handler.py
  run_application    handler.py
  handle_one_response pyw...
  handle_one_request pyws...
  handle             pywsgi.py 464:1
  handle             pywsgi.py 1580:1
  _handle_and_close_when_done
```

The flask_socketio library uses socketio to handle websocket connections. The socketio library is seen to send polling requests prior to establishing a successful websocket and after said websocket connection fails. However, the actual websocket upgrade request will come in the form of a long-polling GET request that has the header Connection set to upgrade and the header Upgrade set to websocket.

In the flask socketio WSGIServer, it will ultimately respond with a sec-websocket-accept and confirm the websocket connection, after which it will constantly listen on the socket hosting the websocket for frames.

The entry point of the websocket begins after the TCP header parsing.

<https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L694>

handle_one_request() will call handle_one_response() to send back the response that tells the browser that the upgrade is agreed on.

<https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L980>

handle_one_response() initiates the response generation and sending process. It will call run_application() to construct the websocket response and write the response to the client.

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L62>

run_application() will use upgrade_websocket() to generate the websocket response, after storing the response, it will then use write() to send the response to the client, and run_websocket() to have the current socket listen for websocket frames.

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L90>

upgrade_websocket() performs various exception checks to ensure the request is a valid websocket upgrade request. This includes checking the request method to be GET, if the request actually asked for upgrade and the upgrade type.

After passing the conditionals, upgrade_connection() is called.

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L137>

upgrade_connection() uses the environ dict to perform key and version check. It will then prepare the response headers.

upgrade_connection() creates new dict headers and assigns key values for Upgrade, Connection, and more importantly a base64 encoded value that acts as a ack to the upgrade request.

start_response() is then called to begin the final response header construction.

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L243>

calls super.start_response(), which handles additional parsing of header key value pairs into a response header list with latin-1 encoding.

<https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L802>

start_response() validates the headers passed down the parameters, then it returns self.write()

back in start_response(), _prepare_response() is called to assign several constructor values used for upholding websocket connection. It will then return

upgrade_connection returns as well.

upgrade_websocket() returns

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L74>

Back in run_application, write() is called to write the response into the socket.

<https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L766>

write() checks the result of the previous parsing to ensure the response is valid, then it calls _write_with_headers()

<https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L781>

`_write_with_headers()` performs a loop to construct a byte array via appending the already parsed and encoded header key value pairs as well as the necessary `:`, `'` and `'\r\n'`

in `_write_with_headers()`, `_sendall()` is used to send all the headers response and, `_write` is used to send the body of the response.

Afterwards, the chain of calls returns back to `run_application`, where `run_websocket()` is called to start the actual websocket.

<https://github.com/wwwtyro/gevent-websocket/blob/master/geventwebsocket/handler.py#L34>

`run_websocket()` performs a number of calls, the main purpose of these is to actually “upgrade” our socket from polling into websocket. This is again, necessary because flask socketio uses socketio which utilizes long-polling to start the websocket, it also defaults to long-polling if a websocket fails or timeout

A number of `__call__` is used to ultimately get to the while true loop needed for the websocket.

<https://github.com/pallets/flask/blob/main/src/flask/app.py#L2208>

`__call__` of the Flask class is triggered by `list(self.application())` from `run_websocket`, this proceed to another `__call__`

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/_init_.py#L40

Flask.`__call__()` will call `self.wsgi_app`, which triggers `_SocketIoMiddleware's __call__`

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/middleware.py#L45>

We are now in `WSGIApp.__call__`

the chain of `__call__` will call `handle_request()` of socketio, which will call engineio's `handle_request()`.

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/server.py#L323>

`handle_request()` will once again perform validation that the request is a properly formatted websocket upgrade request. It will call `handle_get_request()`.

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/socket.py#L94>

`handle_get_request()` will use `getattr()` to call `_upgrade_websocket` via method names on self, which is a `engineio.socket.Socket` object.

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/socket.py#L150>

`_upgrade_websocket()` will then spawn in a async thread `_websocket_handler` using `__call__`

https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/async_drivers/gevent.py#L33

this calls `self.app()` which will begin `_websocket_handler` that handles websocket frames

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/socket.py#L160>

the `_websocket_handler` object will use a while true loop to constantly receive on the socket

This is also where the frame will be parsed if one has been received. This is achieved by

calling `websocket_wait()`

<https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/socket.py#L162>

`websocket_wait()` will call `WebSocketWSGI.wait()`, check if the frame is valid size, and return it.

https://github.com/miguelgrinberg/python-engineio/blob/main/src/engineio/async_drivers/gevent.py#L52

`WebSocketWSGI.wait()` will simply read from the socket and return the data. The read is done via

<https://github.com/jgelens/gevent-websocket/blob/master/geventwebsocket/websocket.py#L309>

`geventwebsocket.websocket.receive()` will check if the websocket is closed first, and then call `WebSocket.read_message()`

<https://github.com/jgelens/gevent-websocket/blob/master/geventwebsocket/websocket.py#L249>

`read_message()` takes the output of `WebSocket.read_frame()`, which is a tuple of the header and payload of the websocket frame. Afterwards, it will perform conditionals on the opcode and return the payload message.

<https://github.com/jgelens/gevent-websocket/blob/master/geventwebsocket/websocket.py#L193>

`read_frame()` blocks until a entire frame has been read, it will call `decode_header()` to obtain important websocket frame information such as the mask

After `decode_header()`, `read_frame()` will unmask the payload.