Open-Source Report for WebSocket

General Information & Licensing

Code Repository	Flask-socketio / Python
License Type	MIT license
License Description	• 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	• you can't hold the code author(s) legally liable for any reason. • You also can't delete the copyright notice and original license from your version of the code.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ✧ ≈

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

socketio.run(app)
https://github.com/MaRonggg/5bytes/blob/325c16e9d3659d55ecb4abeef3986cdb32cee78a/app.py#L102

https://github.com/MaRonggg/5bytes/blob/325c16e9d3659d55ecb4abeef3986cdb32cee78a/app.py#L25

init()

https://github.com/miguelgrinberg/Flask-SocketIO/blob/91b5ddc31bebeb6241d281252c711b160550ce01/src/flask_socketio/__init__.py#L171

https://github.com/miguelgrinberg/Flask-SocketIO/blob/91b5ddc31bebeb6241d281252c711b160550ce01/src/flask_socketio/__init__.py#L152

The init() will use passing parameters to initialize class variables. In our case, we will pass the 'app' as our only arguments and the function will initialize the rest of variables with their default value.

From library:

param cors_allowed_origins: Origin or list of origins that are allowed to connect to this server. Only the same origin is allowed by default. Set this argument to ``'*'`` to allow all origins, or to ``[]`` to disable CORS handling.

Analysis:

Our project will use SocketIO to wrap around our 'app' to enable the websocket connection.

The default mode of async_mode to threading is being set once we make the function call.

Then, inside the SocketIO, a server is created and tcp_connection is made before the websocket connection.

While initializing the SocketIO, the parameter of 'cors_allowed_origins' will be passed.

This parameter will allow the additional origins to connect to the current server.

As showed in our code, our additional origins will be

socketio = SocketIO(app, cors_allowed_origins=['*', 'http://localhost:8000',

'http://localhost:8080', 'https://localhost', 'https://www.5bytes.org', 'https://147.182.180.28:8080', 'https://147.182.180.28:8000', 'http://www.5bytes.org'])

Next after the init(),

The init_app() will be called.

https://github.com/miguelgrinberg/Flask-SocketIO/blob/91b5ddc31bebeb6241d281252c711b160550ce01/src/flask_socketio/__init__.py#L191

Init_app() will update the 'server_options' and pop out the 'manage_session' from it.

Next,

https://github.com/miguelgrinberg/Flask-SocketIO/blob/91b5ddc31bebeb6241d281252c711b160550ce01/src/flask_socketio/__init__.py#L250

We will set up the wsgi use _SocketIOMiddleware() while passing the server and app as arguments.

The app.wsgi will simply allow us to expose the Flask application in the WSGI environment before executing the request

Since we have the parameter of 'app' when creating the SocketIO.

At this point, we set up the socket correctly, we will turn it on and make sure it will get the message for our needs.

@socketio.on()

https://github.com/MaRonggg/5bytes/blob/325c16e9d3659d55ecb4abeef3986cdb32cee78a/app.py#L31

on()

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L258

decorator(handler)

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L279

self.handlers.append((message, _handler, namespace))

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L288

Our socketio.on enter into on(). And then on() will call the decorator(). Since self.server is currently none, it will go to branch for self.handlers.append((message, _handler, namespace)). The handlers is a [] from SocketIO.__init__(). Therefore decorator() is actually

append tuple(message, _handler, namespace) into handlers.

At this point, we have the handler set up as we did in our homework. The server side will call this handler and use it to handle the coming requests.

run()
https://github.com/miguelgrinberg/Flask-SocketIO/blob/91b5ddc31bebeb6241d281252c711b160550ce01/src/flask_socketio/__init__.py#L553

elif self.server.eio.async_mode == 'eventlet':

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L653

run_server()

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L685

eventlet.wsgi.server(eventlet_socket, app, log_output=log_output, **kwargs)

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L679

Analysis: secketio.run uses Eventlet Networking Library, since we have to force it to use WebSockets instead of polling / long-polling. With calling secketio.run with parameter app, secketio.run goes to branch “elif self.server.eio.async_mode == 'eventlet': ”.

Since we did not set value for use_reloader, run_server() will be called. And then, eventlet.wsgi.server(eventlet_socket, app, log_output=log_output, **kwargs) will make the websocket connection.