Flask-Sock

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/flask-sock
License Type	MIT License
License Description	● With copyright included: Copyright © 2021 Miguel Grinberg ● Permission granted free of charge to any person ● Rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
License Restrictions	● No liability ● No warranty ●
Who worked with this?	James Aquilina, Will Marchant, Ryan Yam, Daniil Khan

Sock()

Purpose

This tech allows us to run our server using flask (report already provided for flask) while giving us the ability to utilize WebSockets in addition to the base flask server. Websockets allow a two-way connection between the server and our clients. We can use them to push new data to clients without sending new HTTP requests which enables real-time communication between the client and server. The call for Sock() can be found in the __init__.py file on line 18.

Magic

- The technology allows us to open a WebSocket connection for every client to the server after the TCP socket is created. With this WebSocket connection, we can send frames between the client and server to update the client in real-time. This allows for features such as instant private messaging and changing when a user is considered online or not. The server can receive a WebSocket frame through the connection from the client and will read the information needed then send a frame back to the client to determine what is updated on the client side.
- After importing the flask_sock library and creating a WebSocket connection with our app in __init__.py we can call __.receive() eventually to receive frames.
- **First Link:** Sock is a class from the flask_sock library. It creates a Sock() object that acts as the core of the WebSocket connection. It takes no parameters to create the object and sets most of the values to 'None' when initialized. https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L6
- **Second Link:** If the value of 'app' is 'None' when the Sock object is initialized, then the object sets the value of 'bp' to Blueprint('__flask_sock', __name__). This sets the blueprint value of the Sock object to a Blueprint that acts similar to addRoute(Route Object) that Jesse used in his homework solutions. Which provides the action/function when a route is requested. The full report for the blueprint can be found in the report on the flask framework. https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L16
- **Third Link:** The Blueprint function comes from the flask library and is specifically from the 'blueprints.py' file on line 112 in the flask library. https://github.com/pallets/flask/blob/a25d234cdddf584528dd6af21db736b6faad27af/src/flask/blueprints.py#L112
- **Fourth Link:** This blueprint function takes a Scaffold object which can be found in the 'scaffold.py' file on line 61 in the flask library. This Scaffold is basically the structure behind the blueprint and other items in flask. https://github.com/pallets/flask/blob/a25d234cdddf584528dd6af21db736b6faad27af/src/flask/scaffold.py#L61
- **Fifth Link:** Going back to the Sock() function itself, it also calls the 'init_app(self, app)' function. This function calls the register_blueprint function on app to register a class onto the blueprint being provided. https://github.com/pallets/flask/blob/a25d234cdddf584528dd6af21db736b6faad27af/src/flask/blueprints.py#L254
- After the blueprint is registered, the Sock object is initialized, and should be open to receive WebSocket connections from clients to the server.

.init_app(self, app)

Purpose

Init_app is a function that is used simply for the purpose of initializing the flask-sock extension. It creates a blueprint for the Sock object that will be used similarly to the 'Route' class discussed in our 312 class. It takes a parameter 'app', which is created by the flask application. This is discussed further in the report about flask as a library. The blueprint that this function sets helps in the future for creating a WebSocket connection due to the blueprint acting similar to a 'Route' class. This tech can be found on line 39 in __init__.py in our project and is technically situational (this will be further explained later).

Magic

- This technology is an extension of the Sock object mentioned above in this report. This function allows for a blueprint set in Sock's 'bp' value to be registered to the flask application it is working with. If the 'app' parameter of the Sock instantiation is None then this function will be called, however, if the app parameter provides a correct flask application, this function will not be called and is situational to that instance. Nonetheless, for our project, this technology was used and its operations will be explained further in detail below.
- **First Link:** As mentioned before, the call for this technology is situational, in our case, we did not specify a flask application in the original Sock instantiation. Therefore, the init_app is called and registers a blueprint that is instantiated from Sock(). https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L19
- **Second Link:** When this is called, it runs the init_app() function which registers a blueprint. This register_blueprint is located in the flask library under blueprint.py, in this case, it registers the blueprint that is made by Sock() which adds it to a list of blueprints to be used when the operation is needed. https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L30
- **Third Link:** In this register_blueprint function, the blueprint to be registered is taken as a parameter along with any options if needed. The function throws a ValueError exception if the blueprint is trying to register a blueprint on itself. This is a built-in exception and will be referenced in the fourth link. When a blueprint is registered, it is effectively added to a list of blueprints in a blueprint object that will be used when needed. https://github.com/pallets/flask/blob/fac630379d31baaa1667894c2b5613304285a4bb/src/flask/blueprints.py#L254
- **Fourth Link:** The last link will be a built-in exception error known as ValueError. This exception is thrown when a function receives a parameter that has the right type but not the right value. Basically, if I have a class called webapp that has an __init__ function that has a list of webapps, it will throw a ValueError if I attempt to append that webapp I made to itself. This can be due to the simple fact that something cannot contain itself. https://docs.python.org/3/library/exceptions.html#:~:text=exception-,ValueError,-%C2%B6

.receive()

Purpose

This tech receives WebSocket frames for the server to parse. It can be seen as similar to the .recv function in the socket server library. It is needed for live updating of the client and is separate from sending a request for a path. This data contains what is sent from the client to the server and eventually will have data sent back to the client. Receiving as a function is essential to WebSocket connections being able to work.It is used on line 52 of __init__.py in our project.

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 for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
- 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 technology allows for WebSocket frames to be sent to the server and received to be parsed and used for the data in the frames. This is similar to the .recv function in the socket server library which receives data sent from a client to the server. This function allows for data to be received over the WebSocket connection which is different from receiving requests from a client. This data is not a request and is just raw bytes that contain information prevalent to the client-side of a web application. This can involve a live chat, video chat, and much more.
- **First Link:** The first portion of this receive resides in the simple-websockets library. The call is never explicitly used in the flask-sock library, however it imports the Server from the simple-websockets library which utilizes receive in the route function of flask-sock.https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L32
- **Second Link:** The flask-sock library utilizes the Server class in the ws.py file in the simple-websockets library. The instantiation of the Server object establishes the environment of the app and checks if a socket connection is valid. Which allows the reception of data later in the program. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072

d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L209

- **Third Link:** This links to the ws class in the ws.py file. This helps to create a websocket connection that can send and receive data. Due to the nature of the class and it's responsibilities it contains a send and receive function which work on receiving and sending information across the established websocket connections. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L39
- **Fourth Link:** The receive function of this ws object takes a parameter called timeout. This tells the websocket connection how long it should wait for new data to come in, and if it exceeds that amount of time to timeout. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L95
- **Fifth Link:** This function looks out for two things, if the socket is connected and if the buffer has any data. This essentially tells the websocket connection whether or not there is a timeout needed because of a lack of data being received. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L105
- **Sixth Link:** If the buffer is receiving data and the socket is connected then the buffer sends the information to the server and removes it from the buffer array. In the case that the socket is not connected it would call the ConnectionClosed function. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L109
- **Seventh Link:** The ConnectionClosed function sends a reason for the websocket connection to be closed. In this case, it will be for a timeout disconnect. This gives the server a message that the connection has been terminated. https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L109

.send()

Purpose

This tech helps to send a WebSocket frame to the client from the server. This replaces the need to create a websocket frame from scratch and send it through the socket server. The send function serves an important role in ensuring data on the client can be updated in real time. The sending of these frames allows for features such as live messaging, active display changes, and much more. Without these frames, a lot of features of the web application would not work and would require a refresh to send a separate request to update the page. This would create inconvenience and discourage users from utilizing the features provided by the web app. ws.send() is used on line 147 in __init__.py.

Magic

- **The first three links are the same as receive, due to similar process.**
  First Link: The first portion of this receive resides in the simple-websockets library. The call is never explicitly used in the flask-sock library, however it imports the Server from the simple-websockets library which utilizes receive in the route function of flask-sock.
  https://github.com/miguelgrinberg/flask-sock/blob/a390d5705537b0cb127280fe2a41b59e92990d66/src/flask_sock/__init__.py#L32
- **Second Link:** The flask-sock library utilizes the Server class in the ws.py file in the simple-websockets library. The instantitation of the Server object establishes the environment of the app and checks if a socket connection is valid. Which allows the reception of data later in the program.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L209
- **Third Link:** This links to the ws class in the ws.py file. This helps to create a websocket connection that can send and receive data. Due to the nature of the class and it's responsibilities it contains a send and receive function which work on receiving and sending information across the established websocket connections.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L39
- **Fourth Link:** Send is the function in focus for this portion of the report. The send function in simple-websockets under the ws.py file takes a parameter 'data'. This will be the data to send to the client and can be encoded in bytes or be a simple string. Like previously mentioned, this function allows for less complicated data sending by creating the frame for us and sending it.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L80
- **Fifth Link:** The function ensures that the connection is valid first and if it is not it calls the ConnectionClosed function.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L87
- **Sixth Link:** The ConnectionClosed function sends a reason for the websocket connection to be closed. In this case, it will be that the connection is not valid and cannot send anything over an invalid connection. This gives the server a message that the connection has been terminated.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L109
- **Seventh Link:** The next portion of the send code ensures that the data is bytes or a string. If it is in bytes it will instantiate a Message object and send that object.
  https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L89
- **Eight Link:** The message object is located in the wspronto library in the events.py file. This object sets the data and establishes if the frame is created and if the message is in the frame. This is the frame that will be sent to the client.
  https://github.com/python-hyper/wsproto/blob/4580e161c88bfbbaae5bf5fc0baefcc79ab45535/src/wsproto/events.py#L193
- **Ninth Link:** If the data is not bytes, the send function will create a TextMessage object and send that as the frame instead. This is also located in the wspronto library and instead just stores the data as a string in the object to be sent.
  https://github.com/python-hyper/wsproto/blob/4580e161c88bfbbaae5bf5fc0baefcc79ab45535/src/wsproto/events.py#L224
- **Tenth Link:** The send function will create these frames and store them as out_data. This out_data will be sent over the socket which is the last step in sending the frames over the websocket. It uses the socket framework which

operates closely to the socketserver framework we learned in class.
https://github.com/miguelgrinberg/simple-websocket/blob/9e024fdbb0c9eb6fe8072d8ef05aa6393f1fbc87/src/simple_websocket/ws.py#L93