Open-Source Report

Proof of knowing your stuff in CSE312

Flask (Python)

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/Flask-SocketIO
License Type	The MIT License (MIT)
License Description	Copyright (c) 2014 Miguel Grinberg 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:
License Restrictions	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.

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? 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

```
socketio = SocketIO(app)
```

SocketIO():

- This class can be found in **Flask-SocketIO/src/flask_socketio/__init__.py** on line **54**

- This class is used to create and initialize the socketIO server. The input will be the flask application and can be used to initiate a websocket communication. Using the initialized socket, several different methods can be used including on() which is used to register a socketIO event handler (similar to @app.route()).

```
54  class SocketIO(object):
55      """Create a Flask-SocketIO server.
56
57      :param app: The flask application instance. If the application instance
58                  isn't known at the time this class is instantiated, then call
59                  ``socketio.init_app(app)`` once the application instance is
60                  available.
61      :param manage_session: If set to ``True``, this extension manages the user
62                             session for Socket.IO events. If set to ``False``,
63                             Flask's own session management is used. When using
64                             Flask's cookie based sessions it is recommended that
65                             you leave this set to the default of ``True``. When
66                             using server-side sessions, a ``False`` setting
67                             enables sharing the user session between HTTP routes
68                             and Socket.IO events.
69      :param message_queue: A connection URL for a message queue service the
70                            server can use for multi-process communication. A
71                            message queue is not required when using a single
72                            server process.
73      :param channel: The channel name, when using a message queue. If a channel
74                      isn't specified, a default channel will be used. If
75                      multiple clusters of SocketIO processes need to use the
76                      same message queue without interfering with each other,
77                      then each cluster should use a different channel.
78      :param path: The path where the Socket.IO server is exposed. Defaults to
79                   ``'socket.io'``. Leave this as is unless you know what you are
80                   doing.
81      :param resource: Alias to ``path``.
82      :param kwargs: Socket.IO and Engine.IO server options.
```

```
@app.route('/auctions')
```

route():

- route(‘/auctions’) function within the scaffold class in **flask/src/flask/scaffold.py** , line **423** takes in the added URL as an input. An extra method can be added to the function to be executed once the URL is received.

- The code will load the auctions.html file, which will upon initialization perform the websocket handshake and be left open to receive any incoming messages.

```
@socketio.on('replace old value')
```

socketio.on():

- This class can be found in **Flask-SocketIO/src/flask_socketio/__init__.py** within the **SocketIO()** class on line **258**

- This method takes in the name of a certain event (depending on what you name your incoming socket request) and executes the code defined in a method of your choice created under this method. Although not used in our project, there is also the option to add a second parameter, which can be used as a namespace that specifies the handler in which it should be registered. This will default to ‘/’ if none is specified

```python
def on(self, message, namespace=None):
    """Decorator to register a SocketIO event handler.

    This decorator must be applied to SocketIO event handlers. Example::

        @socketio.on('my event', namespace='/chat')
        def handle_my_custom_event(json):
            print('received json: ' + str(json))

    :param message: The name of the event. This is normally a user defined
                    string, but a few event names are already defined. Use
                    ``'message'`` to define a handler that takes a string
                    payload, ``'json'`` to define a handler that takes a
                    JSON blob payload, ``'connect'`` or ``'disconnect'``
                    to create handlers for connection and disconnection
                    events.
    :param namespace: The namespace on which the handler is to be
                      registered. Defaults to the global namespace.
    """
    namespace = namespace or '/'

    def decorator(handler):
        @wraps(handler)
        def _handler(sid, *args):
            return self._handle_event(handler, message, namespace, sid,
                                      *args)

        if self.server:
            self.server.on(message, _handler, namespace=namespace)
        else:
            self.handlers.append((message, _handler, namespace))
        return handler
    return decorator
```

```python
emit('replaced values', message, broadcast=True)
```

socketio.emit():

- This class can be found in **Flask-SocketIO/src/flask_socketio/__init__.py** within the **SocketIO()** class on line **401**

- This method is used to send out the messages that are received to the connected clients. In the case of our code, when the event ‘replaced values’ is sent, the message associated with that event is sent along with it and is broadcasted to the connected clients.

```
    def emit(self, event, *args, **kwargs):
        """Emit a server generated SocketIO event.

        This function emits a SocketIO event to one or more connected clients.
        A JSON blob can be attached to the event as payload. This function can
        be used outside of a SocketIO event context, so it is appropriate to
        use when the server is the originator of an event, outside of any
        client context, such as in a regular HTTP request handler or a
        background task. Example::

            @app.route('/ping')
            def ping():
                socketio.emit('ping event', {'data': 42}, namespace='/chat')

        :param event: The name of the user event to emit.
        :param args: A dictionary with the JSON data to send as payload.
        :param namespace: The namespace under which the message is to be sent.
                          Defaults to the global namespace.
        :param to: Send the message to all the users in the given room, or to
                   the user with the given session ID. If this parameter is not
                   included, the event is sent to all connected users.
        :param include_self: ``True`` to include the sender when broadcasting
                             or addressing a room, or ``False`` to send to
                             everyone but the sender.
        :param skip_sid: The session id of a client to ignore when broadcasting
                         or addressing a room. This is typically set to the
                         originator of the message, so that everyone except
                         that client receive the message. To skip multiple sids
                         pass a list.
        :param callback: If given, this function will be called to acknowledge
                         that the client has received the message. The
                         arguments that will be passed to the function are
                         those provided by the client. Callback functions can
                         only be used when addressing an individual client.
```