Open-Source Report

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

[Flask_SOCKETIO]

General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/Flask-SocketIO
License Type	MIT
License Description	• A free software license for a copyrighted work that offers freedoms such as publishing a work to the public domain. • Licensed works, modifications, and larger works may be distributed under different terms and without source code. • Grants use rights, including right to relicense (allows prioritization, license compatibility)
License Restrictions	• It carries only minimal restrictions on how the software can be used, modified, and redistributed, usually including a warranty disclaimer.

	• 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 non infringement. 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. (liability and warranty)

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? 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

For the SocketIO library in Flask frame, it is used to establish Websocket connections. Basically, when the SocketIO library is called, it is allowed by the Flask application to establish a low latency bi-directional communications between the clients and the server, which means that it can establish Websocket connection. The line 25 shows in the below image shows that how this library work in our code:

app.py __init__.py

app.py > ...

```python
from flask import Flask, render_template, request
from flask_socketio import SocketIO, send, emit
import json
from pymongo import MongoClient
import cookie_engine
import random
import os

import game_engine


mongo_client = MongoClient("mongo")
db = mongo_client["proj"]
users_info_collection = db["users_info"]

users_account = db["users_account"]
users_test_account = db["users_account"]
cookies_collection = db["cookies_collection"]
game_collection = db["game_collection"]

#dir_path = os.getcwd()
#app = Flask(__name__, static_url_path="/static", template_folder= dir_path)
app = Flask(__name__, static_url_path="/static")
app.config['SECRET'] = "secret!123"
socketio = SocketIO(app, cors_allowed_origins="*")

@app.route('/')
def index():  # put application's code here
    # users_account.drop()
    return render_template("lobby.html")

```

In line 25, we run the SocketIO with two arguments, which are app and cors_allowed_origins, and assign the value to the variable socketio. After this line is executed, it will create a Flask-socketio server, which also establishes a websocket connection. The library code is shown between line 54 and line 169. The screenshot of library is here (line 54 - line 169) in /src/flask_socketio/__init__.py (https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py):

```
class SocketIO(object):
    """Create a Flask-SocketIO server.

    :param app: The flask application instance. If the application instance
                isn't known at the time this class is instantiated, then call
                ``socketio.init_app(app)`` once the application instance is
                available.
    :param manage_session: If set to ``True``, this extension manages the user
                           session for Socket.IO events. If set to ``False``,
                           Flask's own session management is used. When using
                           Flask's cookie based sessions it is recommended that
                           you leave this set to the default of ``True``. When
                           using server-side sessions, a ``False`` setting
                           enables sharing the user session between HTTP routes
                           and Socket.IO events.
    :param message_queue: A connection URL for a message queue service the
                          server can use for multi-process communication. A
                          message queue is not required when using a single
                          server process.
    :param channel: The channel name, when using a message queue. If a channel
                    isn't specified, a default channel will be used. If
                    multiple clusters of SocketIO processes need to use the
                    same message queue without interfering with each other,
                    then each cluster should use a different channel.
    :param path: The path where the Socket.IO server is exposed. Defaults to
                 ``'socket.io'``. Leave this as is unless you know what you are
                 doing.
    :param resource: Alias to ``path``.
    :param kwargs: Socket.IO and Engine.IO server options.

    The Socket.IO server options are detailed below:

    :param client_manager: The client manager instance that will manage the
                           client list. When this is omitted, the client list
                           is stored in an in-memory structure, so the use of
                           multiple connected servers is not possible. In most
                           cases, this argument does not need to be set
                           explicitly.
    :param logger: To enable logging set to ``True`` or pass a logger object to
                   use. To disable logging set to ``False``. The default is
                   ``False``. Note that fatal errors will be logged even when
                   ``logger`` is ``False``.
```

```
    :param json: An alternative json module to use for encoding and decoding
                 packets. Custom json modules must have ``dumps`` and ``loads``
                 functions that are compatible with the standard library
                 versions. To use the same json encoder and decoder as a Flask
                 application, use ``flask.json``.
    :param async_handlers: If set to ``True``, event handlers for a client are
                           executed in separate threads. To run handlers for a
                           client synchronously, set to ``False``. The default
                           is ``True``.
    :param always_connect: When set to ``False``, new connections are
                           provisory until the connect handler returns
                           something other than ``False``, at which point they
                           are accepted. When set to ``True``, connections are
                           immediately accepted, and then if the connect
                           handler returns ``False`` a disconnect is issued.
                           Set to ``True`` if you need to emit events from the
                           connect handler and your client is confused when it
                           receives events before the connection acceptance.
                           In any other case use the default of ``False``.

    The Engine.IO server configuration supports the following settings:

    :param async_mode: The asynchronous model to use. See the Deployment
                       section in the documentation for a description of the
                       available options. Valid async modes are ``threading``,
                       ``eventlet``, ``gevent`` and ``gevent_uwsgi``. If this
                       argument is not given, ``eventlet`` is tried first, then
                       ``gevent_uwsgi``, then ``gevent``, and finally
                       ``threading``. The first async mode that has all its
                       dependencies installed is then one that is chosen.
    :param ping_interval: The interval in seconds at which the server pings
                          the client. The default is 25 seconds. For advanced
                          control, a two element tuple can be given, where
                          the first number is the ping interval and the second
                          is a grace period added by the server.
    :param ping_timeout: The time in seconds that the client waits for the
                         server to respond before disconnecting. The default
                         is 5 seconds.
    :param max_http_buffer_size: The maximum size of a message when using the
                                 polling transport. The default is 1,000,000
                                 bytes.
    :param allow_upgrades: Whether to allow transport upgrades or not. The
                           default is ``True``.
```

```
    :param http_compression: Whether to compress packages when using the
                             polling transport. The default is ``True``.
    :param compression_threshold: Only compress messages when their byte size
                                  is greater than this value. The default is
                                  1024 bytes.
    :param cookie: If set to a string, it is the name of the HTTP cookie the
                   server sends back to the client containing the client
                   session id. If set to a dictionary, the ``'name'`` key
                   contains the cookie name and other keys define cookie
                   attributes, where the value of each attribute can be a
                   string, a callable with no arguments, or a boolean. If set
                   to ``None`` (the default), a cookie is not sent to the
                   client.
    :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.
    :param cors_credentials: Whether credentials (cookies, authentication) are
                             allowed in requests to this server. The default is
                             ``True``.
    :param monitor_clients: If set to ``True``, a background task will ensure
                            inactive clients are closed. Set to ``False`` to
                            disable the monitoring task (not recommended). The
                            default is ``True``.
    :param engineio_logger: To enable Engine.IO logging set to ``True`` or pass
                            a logger object to use. To disable logging set to
                            ``False``. The default is ``False``. Note that
                            fatal errors are logged even when
                            ``engineio_logger`` is ``False``.
    """

```

Then after the websocket is established, it can handle the events from server to client using on function, which in our code shown in the below between line xx and line xx:

```python
@socketio.on("login", namespace="/")
def signup_test(json):
    print("login")
    username = json["username"]
    password = json["password"]
    print("username is: " + username)
    print("password is: " + password)

    # check if the user in db
    exist_user = users_test_account.find_one({"username":username})
    if exist_user == None:
        feedback = {"status": "False", "username": username}
        emit('login',feedback)
    else:
        salt = exist_user["salt"]
        password_se = cookie_engine.disencry(password, salt)
        if password_se != exist_user["password"]:
            feedback = {"status": "False", "username": username}
            emit('login',feedback)
        else:
            feedback = {"status": "True", "username": username}
            emit('login', feedback)
```

The on function is using the established websocket from the Flask-socketio server to handle different events from server to clients. The library code is shown here (line 258 - line 276)
In /src/flask_socketio/__init__.py
(https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py):

```
    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.
        """
```