Live Interaction Server Side:

I am using socket.on, socket.run, and socket.emit.

Flask-SocketIP allows low latency bi-directional communication between client and server. It is compatible with client side SocketIO which is what we use to send player positions from our Game World.

I am using Flask-SocketIO to establish multiple socket connections with my clients. More specifically, I need to establish a connection with a client that's logged in and receive constant information on their position. The Server must also broadcast or emit the position of every logged in client to each client so that they may render them in the Game World. When a client establishes a connection, the server will also emit the position stored in the database which would be their previous location before logging out.

Flask-SocketIO Rooms will also be used to create a Direct Messaging Feature.

How it works:

```
from flask import Flask, render_template

from flask_socketio import SocketIO



app = Flask(__name__)

app.config['SECRET_KEY'] = 'secret!'

socketio = SocketIO(app)



if __name__ == '__main__':

    socketio.run(app)
```

The web server is started calling socketio.run(), replacing app.run() in standard Flask development. The client must receive a page that loads the Socket.IO library to establish a connection.

It will default to localhost and port 5000 if Host and Port are not specified. Next check the server.eio.async_mode. The Engine IO is a transport protocol that enables real-time bidirectional communication between our server and clients.

We are given three asynchronous models. Eventlet is a concurrent networking library that gives us the WebSocket transports. Gevent has long-polling transport but does not have WebSocket support. It must rely on the gevent-websocket package or uWSGI to load our Flask application. Werkzeug can also be used but only for long-polling transport.

Messages are sent and received as events. Flask-SocketIO must register handlers for these events. For our sever we used events that's just had 1 argument, a position JSON.

```
@socketio.on('my event')

def handle_my_custom_event(json):

    print('received json: ' + str(json))
```

We use the emit() function to return messages backs to the client. It will emit the same namespace as the incoming message by default.

It's important that out website broadcast our message so that all logged in clients can see the position of everyone in real-time. To turn a regular emit() to a broadcast we set the optional broadcast argument to True.

```
@socketio.on('my event')

def handle_my_custom_event(data):

    emit('my response', data, broadcast=True)
```

Clients connected to the namespace will receive the message.

Clients can receive messages from rooms they are in but not from other rooms. So, a client can switch between different users and view just their messages with the user.

```
from flask_socketio import join_room, leave_room

@socketio.on('join')
def on_join(data):
    username = data['username']
    room = data['room']
    join_room(room)
    send(username + ' has entered the room.', to=room)
```

```
@socketio.on('leave')
def on_leave(data):
    username = data['username']
    room = data['room']
    leave_room(room)
    send(username + ' has left the room.', to=room)
```

The emit() function sends a message to all the clients in the room. Clients can join or leave any room. When they connect they are given a session ID for the connection.

https://github.com/miguelgrinberg/Flask-SocketIO/blob/master/flask_socketio/__init__.py
https://flask-socketio.readthedocs.io/en/latest/
https://python-engineio.readthedocs.io/en/latest/index.html