Frameworks

Frontend

- Angular 10

Backend

- Flask (uwsgi)

Database

- MongoDB

Hosting

- AWS - EC2 + SSH

https://github.com/talk-with-me
Arkostin
DuncanBark
AlMyPal
tydhoang

Design

- client states:
    - not connected (NC)
    - in queue (Q)
    - connected (C)
- transitions:
    - NC -> Q: client makes API request to get placed in queue, establishes WS connection to be notified when queue over
    - Q -> C: server notifies client that queue is over, and primes client for receiving messages
    - C -> NC: client clicks "leave" or closes tab
    - NC -> C: not allowed
- protocols
    - RTC: socket.io
    - sockets used for client updates from server only
- rtc events
    - socket.io actually handles a lot of metadata
    - S->C: queue complete
    - S->C: message receive/liked message
- restful events
    - join queue
    - message create
    - disconnect
- message objects
    - server generated
        - id
        - room id
        - author id
        - timestamp
    - client provided
        - nonce (for send parity)
        - message content

API Implementation

- REST
  - auth
    - POST /auth
    - return: client_id, client_secret
  - request queue
    - POST /queue
    - Authorization: secret
    - {queueType: vent | listen | talk} (eventually)
  - send message
  - like message
    - POST /likes
    - {message_id, secret}
- WS (client -> server)
  - join room (?)
    - emit('joinRoom', user_id: string, secret: int)
  - disconnect (?)
    - (user_id, secret)
- WS (server -> client)
  - notify queue
    - "queue_complete"
    - {user_id: string}
  - relay connection
    - "user_connected" | "user_disconnected"
    - <NODATA>
  - relay message
    - "message"
    - {room_id: string, nonce: string, content: string, id: string, timestamp: datetime}
  - relay message
    - "message_liked"
    - {message_id: string, user_id: string}
  - listen for disconnect
    - "disconnect"
    - <NO DATA>
- DB
  - collections
    - user_collection
      - ip

            - client id
            - client secret
            - ~~is in queue~~
            - queue type
            - entered queue at
        - messages
            - see message schema below
        - rooms
            - id (use mongo built in _id?)
            - users (some user identifier)
    - generate auth
        - user_collection.insert_one(...)
    - enter queue
        - users.update_many(...)
    - exit queue
        - scheduled routine
        - when there is a match
            - get two
            - remove from queue (users.update_many(...))
            - and create a room (rooms.insert_one())
            - and broadcast hey you two get a room
    - join room
        - room.update(push that user into users)
    - send message
        - generate timestamp, id, author id
        - messages.insert(...)
    - leave room (optional)
        - room.update(users pull user)

Data Structures

- message objects
    - server generated
        - id
        - author id
        - timestamp
    - client provided
        - room id
        - nonce (for send parity)
        - message content
- system message objects (user connected/disconnected)
- user objects

- room objects

FE -> FE services -(REST)> Backened/Database -(WS)> FE services -> FE