Streamer App - WS and RTC

This is the documentation for the streamer app. It details the messages that will go between the backend and the application.

All data except for the actual video (and one single message type) goes via WebSockets. This outlines the connection process, the reconnect process, the disconnect process and the messages that will be sent to, and expected from the app.

Connecting to the websocket server

Create a WebSocket object that connects to the server on port 8097. Transmit the token gotten from the sign-up process as a header with the name `sec-websocket-protocol`

Initializing a call

In adition to every message listed here, every message the app sends should contain `"videoSource": "streamer_app"`

Notifying the server

Once the app is ready to transmit video to the system we send a connection notice to the server

```
{
    "type": "new-streamer-connection",
    "groupId": int,
    "name": string,
}
```

- groupId: The group you want to stream video to
- name: The streamers name

The server will reply with the following message

```
{
  "type": "streamer-connection-socket-id",
  "socketId": string
}
```

This SocketId is our ID for this session, and is used as an identifier on subsequent messages.

Now the streamer will appear as a link in the system for the selected group.

Receiving Start Video Request

When an operator clicks the link in their desplay a message will be sent over WebSocket to the app. This message will contain the following:

```
{
    "type": 'streamer-start-video',
    "amkSocketId": string,
    "rtcServer": {
        "stun": string,
        "turn1": {
            "urls": string,
            "username": string,
            "credential": string,
            "credentialType": string
        },
        "turn2": {
            "urls": string,
            "username": string,
            "credential": string,
            "credentialType": string
        }
    },
    "turnCreds": [string, ...]
}
```

The amkSocketId should be saved as it needs to be sent back to the WebSocket-server to route the subsequent messages to the correct client.

With this data in hand, the app should attempt to create an RTC connection to the client. See createRtcConnection.js as a reference for the message types transmitted throughout this process.

Some of the rtc message-types will require a WebSocket-message to transmit the information to the client.

Sending Offer To AMK

The process starts by the app sending an offer to the client. This offer is created by rtc. Once it's generated it should be transmitted over WebSocket

```
{
  "type": "streamer-send-offer-to-amk",
  "amkSocketId": string,
  "offer": The offer from RTC,
  "streamerSocketId": string,
  "groupId": int,
  "cameras": [MediaDeviceInfo],
  "currentDeviceId": string
}
```

- amkSocketId: The socketId we received with the `streamer-start-video` message

- streamerSocketId: The socketId we received with the `streamer-connection-socket-id` message
- cameras: An array of the available cameras on the device. See https://developer.mozilla.org/en-US/docs/Web/API/MediaDeviceInfo

  I'm unsure whether something similar exists for apps. The information we need for each camera is a `label` with a descriptive name, and a unique `deviceId` for each camera. The client will request a camera change based on these deviceIds
- currentDeviceId: The deviceId of the camera currently being used.

Receiving answer from AMK

The client responds with an RTC answer

```
{
  "type": "streamer-receive-answer",
  "amkSocketId": string,
  "answer": The answer from RTC
}
```

Candidates

As well as the offer and answer, which there may be multiple of, depending on the success of the negotiation, there will also be sent candidates back and forth between the app and the client.

If the apps RTC instance generates a candidate, we transmit this to the client.

```
{
  "type": "streamer-send-candidate-to-amk",
  "amkSocketId": string,
  "candidate": The RTC Candidate
}
```

We may also receive a candidate in a similar message

```
{
  "type": "streamer-receive-candidate",
  "amkSocketId": string,
  "candidate": The RTC Candidate
}
```

Reconnecting RTC

If the initial connection fails, or if it falls out and the webSockets of both the app and client are still connected, the app may receive a request to attempt to reconnect.

The client will send the following message:

```
{
  "type": "streamer-receive-reconnect-rtc",
  "amkSocketId": string
}
```

Upon which we create and send a new offer to the client.

Once the video is connected

When the video is streaming, the hard work is over. There are not many messages we will receive after this point.

Changing camera

The client may request a different camera from the app. This request will be sent over the RTC data channel. It will be sent with the following JSON message

```
{
  "type": "select cam",
  "deviceId": string
}
```

- deviceId: One of the deviceIds we provided to the client with our offer.

The app should attempt to switch to the camera with the given deviceId.

If this is successful the app should attempt to replace the videoTrack of all connections with the new camera.

In all cases, even if the deviceId is invalid or the camera switch failed we should send the following message over webSocket:

```
{
  "type": "streamer-update-cameras",
  "streamerSocketId": string,
  "groupId": int,
  "cameras": [MediaDeviceInfo],
  "currentDeviceId": string,
}
```

With cameras and currentDeviceId being the same format as we sent with the offer. If a camera switch failed, we remove this camera from the cameras-list, so we prevent another attempt to switch to an erronous camera.

Location Information

The app should send regular position updates to the service.

```
{
  "type": "streamer-update-location",
  "streamerSocketId": string,
  "groupId": int,
  "latitude": double,
  "longitude": double,
  "heading": double
}
```

Ending a video

Stopping video to a given client

If a given client closes their window we will receive the following message:

```
{
  "type": "streamer-amk-stop-video",
  "amkSocketId": string
}
```

The app should then close the RTC connection for this given amkSocketId.

NOTE: Due to a complication for a given user-class there are some socketIds for which we shouldn't close the connection. If an amkSocketId begins with `ext-` the app should do nothing for this given connection.

Ending the video call

Once the user decides they are done sending video, we should clean up in an orderly fashion.

First close all open RTC connections.

Then close the WebSocket connection.

The server will automatically notify all watchers that this call has ended.