NETWORKING CONEXIÓN A LA SALA

Response

GET /api/media/rooms/producers

CLIENT

status: "success" or "failed",

Creamos el **Device** con los RTPCapabilities, y creamos la

El cliente para crear cada producer,

usando el transport, llama .produce()

El cliente usa cada Producer's ID

para empezar a crear los consumers.

El cliente hace su propio **.consume()** con los parámetros de cada Consumer

Conecto al WebSocket, revisar nuevament

Bajar la información del usuario para la vista, y conectar los nuevos Consumers.

tener los Producers

Se emite evento WebSocker: userJoined

versión local de los **Transports**, con los parámetros de los remotos.

result: {

HTTP

SERVER

El servidor, revisa si se excede la cantidad de Workers. Crea un **Worker** y un **Router.** Crea la **Room** en la base de

El servidor busca las RTPCapabilites, y crea dos WebRTCTransports.

El servidor hace la conexión entre transports.

El servidor crea el **Producer**.

El servidor envía de cada Attendee

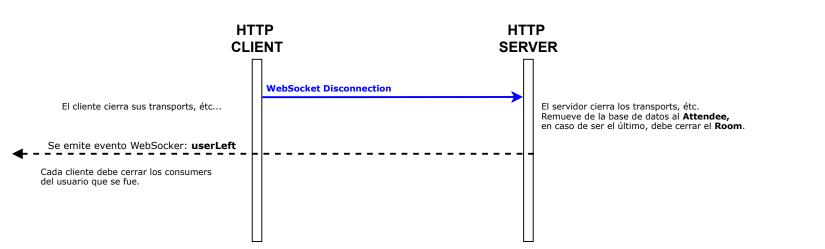
los **Producer's ID** de los streams.

El servidor conecta el transport correspondiente.

El servidor crea cada **Consumer**, y le pasa sus parámetros al cliente. Esto lo hace, ejecutando **.consume()**

El servidor agrega al **Attendee** a la **Room** en la base de datos.

NETWORKING DESCONEXIÓN DE LA SALA

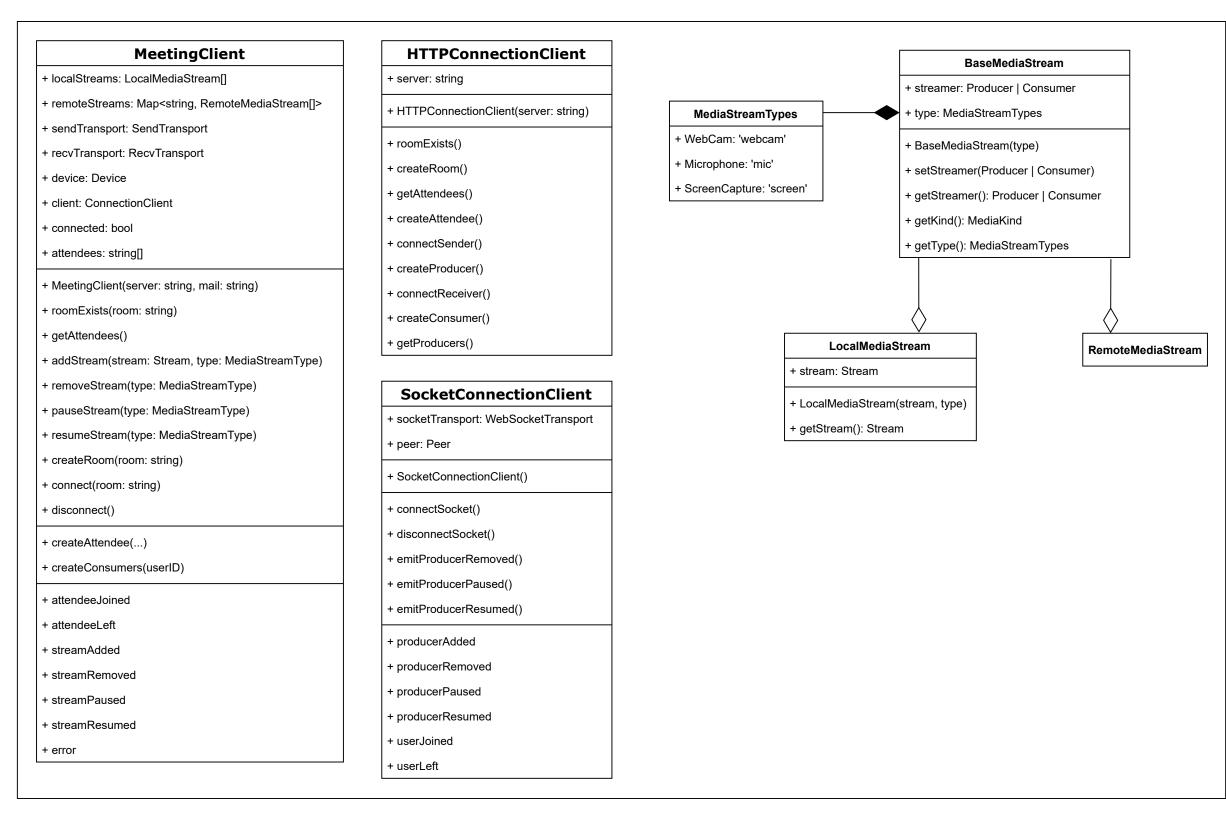


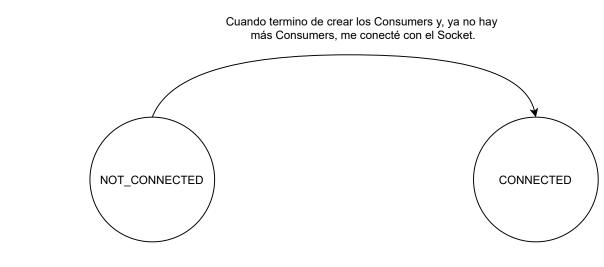
Media HTTP Server	WebSocket Server
OST /api/media/rooms/createRoom	producerAdded: [Server]
OST /api/media/rooms/createAttendee	producerRemoved: [Server, Client]
OST /api/media/rooms/connectSender	producerPaused: [Server, Client]
OST /api/media/rooms/createProducer	producerResumed: [Server, Client]
ET /api/media/rooms/producers	userJoined: [Server]
ST /api/media/rooms/connectReceiver	userLeft: [Server]
OST /api/media/rooms/createConsumer	
ET /api/media/rooms/exists	
/api/media/rooms/attendees	

SERVER API

Attendee Room MeetingServer + id: string + name: string + app: ExpressApp + sendTransport: WebRTCTransport + router: MediaSoupRouter + rooms: Map<string, Room> + recvTransport: WebRTCTransport + room: ProtooRoom + workers: MediaSoupWorker[] + producers: Map<string, Producer> + attendees: Map<string, Attendee> + socketServer: WebSocketServer + consumers: Map<string, Consumer[]> + dispatcher: Map<string, callback> + MeetingServer(database) + Attendee(id) + Room(name, router) + getHTTPRouter() + handleAttendeeConnection() + connectSender() + hasRoom() + handleAttendeeDisconnection() + connectReceiver() + handleAttendeeRequest() + createProducer() RoomSignals + createAttendee() + room: ProtooRoom + createConsumer() + dispatcher: Object + removeProducer() + connectSender() + connectReceiver() + pauseProducer() + RoomSignals() + createProducer() + resumeProducer() + hasAttendee(user: string) + createConsumer() + getProducersInfo() + addAttendee(user: string, transport: WebSocketTransport) + hasAttendee() + getTransportsInfo() + broadcastUserJoined(user: string, producers: List) + getAttendee() + close() + broadcastUserLeft(user: string) + getProducersInfo() + broadcastProducerAdded(user: string, type: MediaStreamType, id: ProducerID) + attendeeJoined + broadcastProducerRemoved(user: string, type: MediaStreamType) + attendeeLeft + broadcastProducerPaused(user: string, type: MediaStreamType) + broadcastProducerResumed(user: string, type: MediaStreamType)

CLIENT API





.addStream(stream): Agrega el MediaStream a la lista. .removeStream(stream): Remueve el MediaStream de la lista. .addStream(stream): Agrega el MediaStream a la lista, y además llama al .produce() .removeStream(stream): Llama al .close(), lo remueve de la lista, emite el evento.