Project

Entwicklung großer Anwendungssysteme

Salvador Jesús Megías Andreu

Matrikel-Nr: 70467293

s.megias@ostfalia.de

Wolfenbüttel, Deutschland - June 22, 2020

Informatik

Contents

Chapter 1

Introducction

The project consists of a basic video conference and text chat system, developed in a basic client-server architecture.

- Based on the *<video>* tag of the *HTML5* standard.
- Based on *https*.
- Real-time data exchange with *WebRTC* (video).
- Use of *Sockets* (*TCP*, connection-oriented).
- Using the *Quarkus* development framework.

1.1 Infraestructure

1.1.1 Server

The operation of the server part is based on the implementation of sockets within the web server included by *Undertow* (*JBoss*).

> A network socket is an internal endpoint for sending or receiving data within a node on a computer network.

The infrastructure on the server side is mainly based on two sockets:

1. **MainSocket** (*wss:localhostmainsocket*). It is a socket to perform the signaling process. It manages the reception and sending of SDP packets and the establishment of the WebRTC call
2. **ChatSocket** (*wss:localhostchatsocket*) . This is a socket for managing users and their messages. Its operation is based on notifying all users of the messages sent by one of them.

1.1.2 Data Exchange Forma

For the data exchange between server and client, for the video socket, we have used today's data exchange standard, JSON.

For this, we had to implement a decoder (to receive the information in JSON format) and an encoder (to send the information in JSON format).

- **Message Model**: *Message.java*
- **Encoder**: *MessageEncoder.java*
- **Decoder**: *MessageDecoder.java*

1.1.3 Client

The interface, is a basic index.html page together with its corresponding css. The magic of its operation is in the index.js.
The signaling server is a necessary server-side app that the client apps use to communicate with one another.

The client creates two WebSockets, one to establish the connection with the text server, and another for the WebRTC 1.1.3 communication management and signaling server..

To sum up, there are four main steps in starting a WebRTC 1.1.3 session:

- The peers need to find their public IP address using a STUN – Session Traversal Utilities for NAT – server.
- After the peers have found their public IP address, client A needs to send 'Offer SDP' to client B through a signaling server, after which client B needs to validate the offer and provide an 'Answer SDP' back to client A. The SDP – Session Description Protocol – is a format for describing streaming media communications parameters that are used to announce a session and to validate its parameters.
- After the session is announced and validated, client A needs to send 'ICE Candidates' to client B through a signaling server, after which client B needs to validate the ICE candidates and reply with the same type of message. The ICE – Interactive Connectivity Establishment – is a method used to retrieve all available candidates (IP addresses), which are then passed to the clients.

- When the session is fully validated and the ICE candidates are ready, the WebRTC 1.1.3 audio and video communication can start, either directly or through a TURN – Traversal Using Relays around NAT – server.

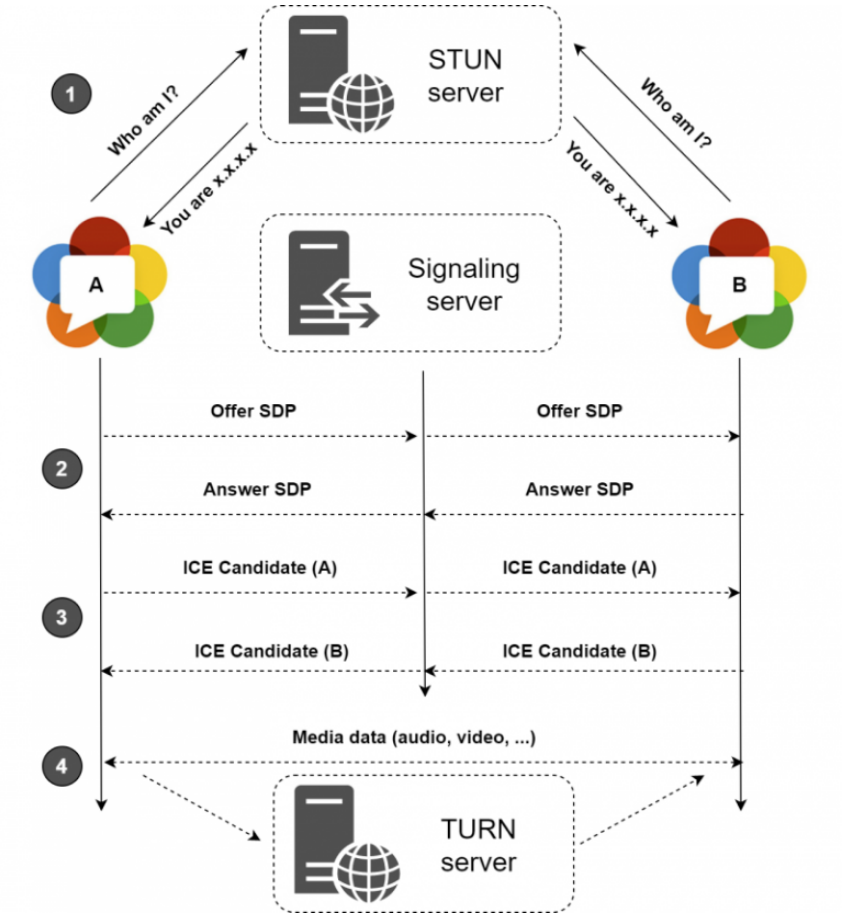

Figure 1.1: Setting up WebRTC connection

Chapter 2

Deployment and operation

2.1 Requirements

- JDK 1.8
- JAVA_HOME well configured.
- Maven installed.
- Internet Connection

2.2 Certificate

In order to enable https communication, a certificate must be displayed (even if it is self-signed). This can be found in META-INF.

In the case of a real deployment, this self-signed certificate should be replaced and the process carried out with certbot or let'sencrypt, but this would require having a valid domain of your own pointing to the server and port 80 open.

In our case, we have made a local deployment, so a self-signed certificate is more than enough.

2.3 Deployment

With quarkus, deploying this project in development mode 2.3 is as simple as that:

```
./mvnw clean compile quarkus:dev
```

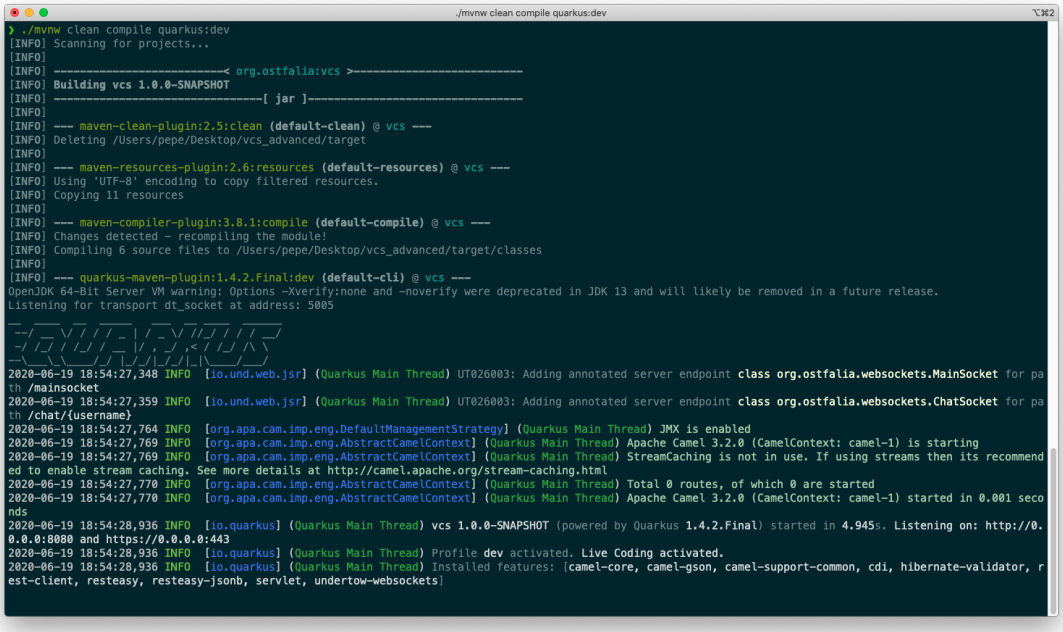

Figure 2.1: Clean and build

2.4 How it works

2.4.1 Desktop

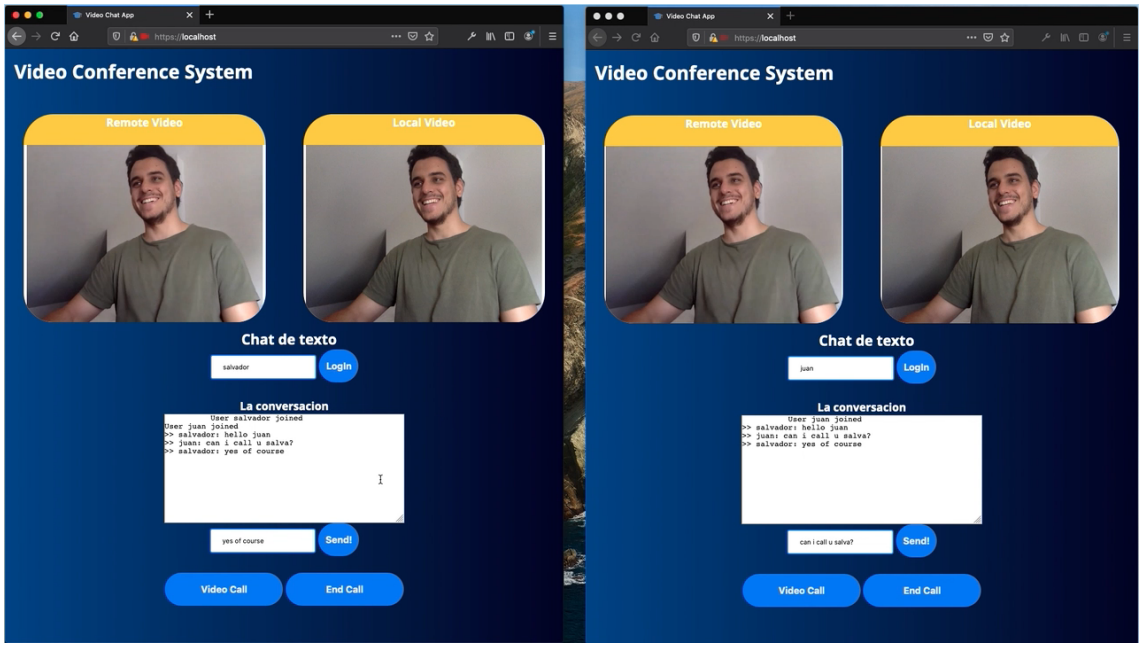

Figure 2.2: Desktop Localhost

https://drive.google.com/open?id=1mPsyT7iQ2b4LJ8W6rEcAzByGkhmsbg_J

2.4.2 Mobile

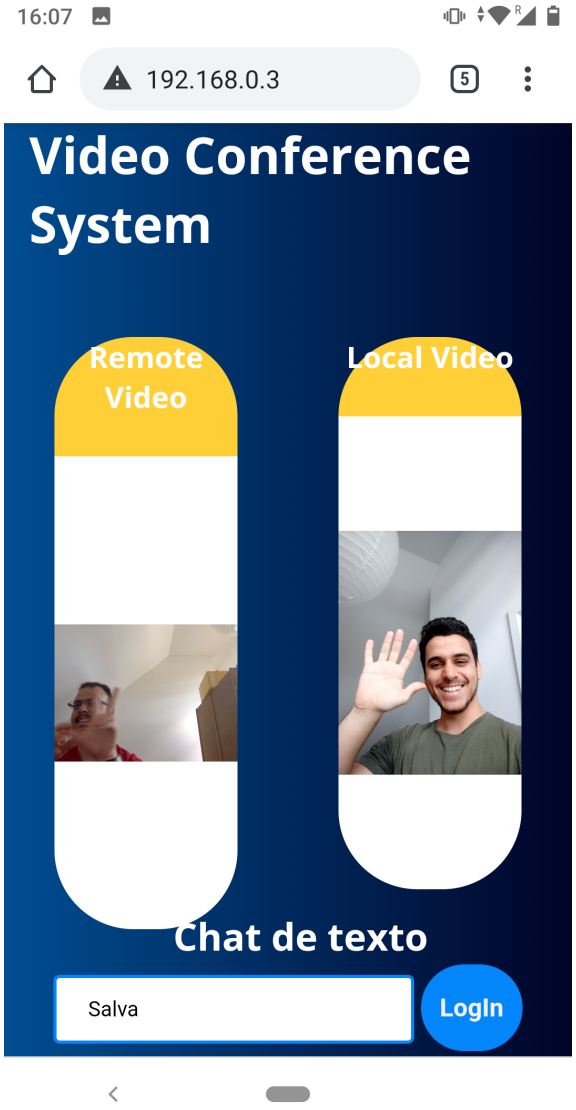

Figure 2.3: Mobile and pc

https://drive.google.com/open?id=1mPsyT7iQ2b4LJ8W6rEcAzByGkhmsbg_J

2.4.3 Exceptional Case

The operation is multiplatform. There is one exception and that is with Safari. Opening a secure socket with a self-signed certificate is not allowed by Safari.

Chapter 3

Difficulties. Future improvements

3.1 Difficulties

Working with a development framework (quarkus) that you don't know, is a great difficulty, but at the beginning. Getting used to it, has been hard, more without knowing other kind of frameworks in depth.

The issue of certificates was unknown to me at first, I had never developed anything for https.

The concept of Socket was unknown to me, but it has been easy to understand and apply.

The most difficult part, by far, is understanding the calling process in WebRTC and the data exchange. It's a simple process once you get to know it, but it's a lot of work at first. In return it is rewarding

I must admit that this project has cost me a gigantic investment of time, but, it has been rewarding and I can understand far beyond, how a service like Skype, or Jitsi can work.

3.2 Future improvements

- The interface is not completely responsive. It would be very reasonable to improve the interface.
- There is no room concept, as there is, for example, in Jitsi, BigBlueButton.

The room concept could imply the scalability of the conference

- There is no user authentication or database. A user database would be great.
- Apply the MVC pattern to the client side. Such an application is difficult to change, debug and maintain.
- Deploy this service in some web-plattform, such as Microsoft Azure, Amazon Web Services.