Construindo uma aplicação de videoconferência com WebRTC - FGA0211 Turma 01

Caetano Santos Lúcio - 1801444979 Guilherme Puida Moreira - 200019015
Irwin Schmitt - 170105342 Thiago Vivan Bastos - 190020407

Introdução

Com o avanço das tecnologias de comunicação em tempo real, os protocolos de camada de aplicação têm desempenhado um papel fundamental na viabilização de serviços como videoconferências, streaming de mídia e colaboração online. Nesse contexto, protocolos como o WebRTC e WebSocket emergem como soluções modernas e eficientes para a troca de dados multimídia e mensagens em tempo real entre navegadores, sem a necessidade de plugins ou softwares adicionais.

O WebRTC (Web Real-Time Communication) é uma tecnologia que permite a transmissão de áudio, vídeo e dados diretamente entre navegadores, facilitando a criação de aplicações de videoconferência com baixa latência. Já o WebSocket é um protocolo que possibilita comunicação bidirecional entre cliente e servidor, o que é essencial para sinalização e controle em tempo real.

O objetivo deste projeto é estudar e compreender a arquitetura desses protocolos e sua aplicação em um ambiente prático. Para isso, será desenvolvida uma aplicação de videoconferência onde múltiplos usuários podem se comunicar por áudio, vídeo e mensagens de texto, permitindo aos usuários criar conferências ou entrar em conferências existentes, proporcionando uma experiência completa de comunicação em tempo real via navegador. A aplicação será dividida em duas frentes, uma *server-side* e outra *client-side* .

O que é WebRTC?

WebRTC (Web Real-Time Communication) é uma tecnologia de código aberto que permite comunicação em tempo real diretamente entre navegadores ou dispositivos móveis. Ele foi desenvolvido para facilitar a transmissão de áudio, vídeo e dados sem necessidade de servidores intermediários para processar a mídia, utilizando um modelo *peer-to-peer*. Isso torna o WebRTC especialmente adequado para aplicações como videoconferências, chamadas VoIP e compartilhamento de arquivos em tempo real. O WebRTC é suportado nativamente pelos principais navegadores modernos, como Chrome, Firefox e Safari.

A grande vantagem do WebRTC é sua capacidade de contornar obstáculos comuns em redes, como NATs (Network Address Translation) e firewalls, para permitir conexões diretas entre pares, garantindo baixa latência. Ele utiliza três componentes principais: `MediaStream`, que lida com a captura e transmissão de áudio e vídeo; `RTCPeerConnection`, que gerencia as conexões *peer-to-peer* e a troca de mídia; e `RTCDataChannel`, que permite a troca de dados arbitrários entre pares, como mensagens de texto ou arquivos.

As principais características do WebRTC são:

1. *Peer-to-peer* direto: A comunicação entre os participantes é feita de forma direta, sem necessidade de um servidor intermediário para transmissão de mídia.
2. Segurança: todo o tráfego WebRTC é criptografado utilizando SRTP (Secure Real-time Transport Procol) para mídia e DTLS (Datagram Transport Layer Security) para o controle.
3. Suporte a múltiplos fluxos de mídia: WebRTC permite a transmissão simultânea de áudio, vídeo e outros tipos de dados.

O ciclo de vida de uma conexão WebRTC pode ser dividido em várias etapas, desde o início da conexão até o seu encerramento. Esse processo envolve a troca de mensagens de sinalização entre os pares e a resolução de problemas

relacionados à descoberta de rotas de comunicação, como NAT traversal. O ciclo de vida típico de uma conexão WebRTC envolve as seguintes fases:

1. Sinalização inicial: O primeiro passo para estabelecer uma conexão WebRTC é a troca de informações de configuração entre os dois pares, conhecida como sinalização. Essa troca geralmente é feita por meio de um servidor intermediário utilizando protocolos como WebSockets ou HTTP. Os navegadores envolvidos trocam informações chamadas de offer e answer, que descrevem as capacidades de mídia e rede de cada lado. O WebRTC não define um protocolo específico para a sinalização, permitindo que os desenvolvedores escolham como realizar essa troca de dados.
2. Criação de RTCPeerConnection: Após a fase de sinalização, cada par cria uma instância de RTCPeerConnection, que é responsável por gerenciar a conexão P2P e a transmissão de mídia. Este objeto lida com a troca de fluxos de áudio e vídeo e também suporta a abertura de canais de dados para a transmissão de dados arbitrários. A RTCPeerConnection também gerencia a criação de ice candidates que são usados para otimizar a rota de comunicação.
3. Troca de ICE Candidates (Interactive Connectivity Establishment): Um dos maiores desafios em uma conexão P2P é garantir que os pares possam se conectar, mesmo quando estão atrás de NATs ou firewalls. O WebRTC utiliza o protocolo ICE para resolver esse problema. Durante essa fase, os pares trocam ice candidates, que são potenciais endereços de rede que podem ser utilizados para estabelecer a conexão. O servidor STUN (Session Traversal Utilities for NAT) é usado para descobrir o endereço público de um dispositivo, enquanto um servidor TURN (Traversal Using Relays around NAT) pode ser usado como intermediário para rotear a mídia caso uma conexão direta não seja possível.
4. Estabelecimento da Conexão: Após a troca de offer, answer e ice candidates, a conexão P2P é finalmente estabelecida. Nesse ponto, os fluxos de mídia começam a ser transmitidos diretamente entre os pares. O WebRTC garante que o áudio e o vídeo sejam sincronizados e transmitidos com a menor latência possível, ajustando dinamicamente a qualidade da mídia conforme as condições da rede variam.
5. Monitoramento da Conexão e Controle de Qualidade: Durante a comunicação, o WebRTC monitora continuamente a qualidade da conexão, utilizando mecanismos como controle de taxa adaptativo e recuperação de pacotes perdidos. Isso é feito através de técnicas como feedback de congestionamento de rede e ajustes automáticos na resolução do vídeo para garantir que a conexão permaneça estável, mesmo em condições adversas de rede.
6. Encerramento da Conexão: Quando um dos pares decide encerrar a chamada ou a comunicação é interrompida por algum motivo (como desconexão da internet), a conexão WebRTC é encerrada. Isso envolve fechar a RTCPeerConnection, o que encerra os fluxos de mídia e dados associados. O encerramento correto da conexão é importante para liberar os recursos de rede e mídia usados pela aplicação.

No contexto do WebRTC, um dos maiores desafios é permitir que os pares se conectem diretamente, mesmo quando estão atrás de NATs (Network Address Translation) ou firewalls, que são comuns em redes domésticas ou corporativas. O WebRTC utiliza o protocolo ICE (Interactive Connectivity Establishment) para resolver esse problema. O ICE é um framework que coordena a descoberta da melhor rota para comunicação entre os pares, combinando o uso de dois servidores: STUN (Session Traversal Utilities for NAT) e TURN (Traversal Using Relays around NAT). O servidor STUN é usado para obter o endereço público e a porta de um cliente por trás de um NAT, permitindo que os pares tentem se conectar diretamente. Quando uma conexão direta não é possível, o WebRTC recorre ao servidor TURN, que atua como um intermediário para retransmitir o tráfego entre os pares. Embora o uso de STUN seja suficiente em muitos casos, o servidor TURN é crucial em situações onde NATs mais restritivos ou firewalls bloqueiam a comunicação direta, garantindo que a conexão seja estabelecida de forma confiável, ainda que com uma ligeira penalidade de desempenho devido ao redirecionamento de tráfego.

Metodologia

Inicialmente, fizemos a implementação de maneira síncrona, com todos os membros do grupo trabalhando em conjunto para definir a estrutura básica e a arquitetura que seria utilizada. Posteriormente, nos comunicamos de maneira assíncrona para incrementar e adicionar novas funcionalidades.

Arquitetura

A aplicação desenvolvida é dividida em duas partes:

1. Uma aplicação *server-side*, responsável por controlar os clientes conectados em cada sala e também propagar as mensagens enviadas por cada um deles.
2. Uma aplicação *client-side*, que é acessada diretamente pelo usuário e permite entrar em salas e se comunicar com outros clientes em uma sala por meio de áudio e vídeo.

O fluxo básico de execução é:

1. O usuário acessa a URL da aplicação.
2. O usuário escolhe uma sala (definida por um idenfificador arbitrário).
3. O cliente notifica ao servidor que o usuário entrou na sala.
4. Caso já existam clientes na sala, eles são notificados que um novo cliente se conectou.
5. Todos os clientes já conectados notificam o novo cliente que desejam estabelecer uma conexão WebRTC.
6. A conexão WebRTC é estabelecida entre o novo cliente e todos os clientes já conectados.

Portanto, um cliente se conecta diretamente com todos os outros clientes presentes na sala em modelo *peer-to-peer*. Após a conexão ser estabelecida, o usuário pode se comunicar com áudio e vídeo, mandar mensagens de texto, controlar o estado de transmissão do áudio e vídeo e compartilhar a tela.

A aplicação *server-side*

A aplicação *server-side* for implementada em TypeScript no runtime Deno, que oferece suporte nativo a WebSockets e uma arquitetura moderna para lidar com código assíncrono e não bloqueante. O servidor é responsável por gerenciar as conexões WebSocket entre os clientes e a sinalização necessária para estabelecer as conexões WebRTC entre eles. Ele atua como um facilitador, coordenando a troca de mensagens de sinalização (*offers*, *answers* e *ice candidates*), mas não participa diretamente da transmissão de mídia, que é feita de forma *peer-to-peer*.

Ao iniciar, o servidor abre um canal WebSocket e fica aguardando conexões dos clientes. Cada cliente que se conecta é identificado por um `uid` único, que é utilizado para gerenciar as sessões e distribuir as mensagens entre os participantes de uma mesma sala. Quando um novo cliente se junta a uma sala, ele envia uma mensagem via WebSocket para o servidor, que, por sua vez, propaga essa mensagem para os outros clientes já conectados na mesma sala. Essa propagação é feita de forma eficiente e assíncrona, garantindo que a sinalização chegue a todos os clientes com a menor latência possível.

O servidor mantém um estado interno simples, utilizando um mapa que associa cada sala a uma lista de *uids* dos clientes conectados. Quando um cliente sai ou se desconecta, ele é removido desse mapa, e o servidor envia uma notificação para os outros participantes da sala, informando sobre a saída do cliente. Isso permite que os clientes ajustem suas conexões WebRTC e encerrem as conexões *peer-to-peer* de forma apropriada. A escolha de utilizar WebSockets para essa sinalização se deve à sua capacidade de suportar comunicação bidirecional em tempo real, permitindo que o servidor envia atualizações para os clientes sempre que necessário, sem depender de requisições periódicas (*polling*).

Além de coordenar a sinalização, o servidor também lida com alguns casos especiais, como a retransmissão de mensagens de candidatos ICE (Interactive Connectivity Establishment). Isso é crucial para garantir que os clientes consigam estabelecer e manter as conexões WebRTC, mesmo em cenários onde a rede apresenta desafios, como NATs e *firewalls*. O servidor recebe os *ice candidates* de cada cliente e os retransmite para os outros, facilitando o processo de descoberta da melhor rota de comunicação entre os pares. Como o WebRTC exige a troca precisa dessas informações para garantir a qualidade da conexão, o papel do servidor nessa etapa é essencial para o funcionamento da aplicação.

A aplicação *client-side*

A aplicação *client-side* é responsável por gerenciar a interface de usuário e estabelecer as conexões WebRTC para permitir a comunicação em tempo real entre os participantes de uma sala. Ela é escrita em JavaScript, rodando diretamente no navegador dos usuários, e faz uso das APIs nativas de WebRTC e WebSocket para implementar as funcionalidades de videoconferência e troca de mensagens de texto.

Quando o usuário entra em uma sala, o cliente cria uma conexão WebSocket com o servidor para gerenciar a sinalização necessária para as comunicações WebRTC. O cliente gera um identificador único (`uid`) e envia uma mensagem para o servidor, informando que seu usuário entrou na sala. Caso já existam outros participantes na sala, o servidor notificará esses clientes para que iniciem o processo de estabelecimento de conexões *peer-to-peer*

com o novo participante. Cada cliente cria uma instância de `RTCPeerConnection` para cada outro cliente na sala, utilizando a API de WebRTC para gerenciar a troca de áudio e vídeo diretamente entre os navegadores.

A parte central da aplicação *client-side* envolve a troca de mensagens de sinalização, como *offers*, *answers* e *ice candidates*. Quando um cliente deseja iniciar uma comunicação WebRTC, ele gera uma *offer* contendo informações sobre as capacidades de mídia (por exemplo, codecs de áudio e vídeo suportados) e envia essa *offer* ao servidor, que propaga para os outros clientes na sala. Esses outros clientes, por sua vez, respondem com uma *answer*, e essa troca permite que as conexões sejam estabelecidas. Além disso, à medida que a conexão vai se desenvolvendo, os clientes trocam mensagens de *ice candidates* para otimizar as rotas de comunicação e garantir a menor latência possível na transmissão de mídia.

Uma das funcionalidades mais importantes implementadas no *client-side* é o controle da mídia local. O cliente usa as APIs do WebRTC para capturar a câmera e o microfone do usuário, e esses fluxos de mídia são transmitidos diretamente para os outros clientes através da conexão *peer-to-peer*. A interface permite que o usuário tenha total controle sobre sua mídia, como silenciar o microfone, desligar a câmera ou compartilhar a tela com outros participantes da sala. Essas funcionalidades são acessíveis através de botões na interface após se juntar à uma sala.

Além da tranmissão de áudio e vídeo, o cliente também permite o envio de mensagens de texto através de um chat embutido, que usa a conexão WebSocket para enviar e receber mensagens em tempo real. O gerenciamento de estados e eventos do WebRTC é implementado no lado do cliente, usando *event listeners*. Tomamos cuidado especial para seguir cuidadosamente o ciclo de vida de uma conexão WebRTC, para lidar bem com os eventos de conexão e desconexão.

WebSockets

A tecnologia de WebSockets tem papel fundamental na nossa aplicação, já que todas as mensagens trocadas entre os clientes são feitas por meio de uma conexão WebSocket entre os cliente e o servidor. A capacidade de comunicação bidirecional e em tempo real facilitou a implementação e permite que o uso de recursos do servidor seja extremamente baixo.

Conclusão

A aplicação desenvolvida é funcional, e atende os requisitos apresentados na definição do trabalho. Além dos requisitos adicionais, conseguimos implementar também funcionalidades comuns em outros sistemas de videoconferência, como compartilhamento de tela e mensagens de texto. Uma limitação atual é a falta de segurança, já que não existe nenhum mecanismo de autenticação e grande parte da funcionalidade é implementada pelo cliente, que pode ser facilmente manipulado pelos usuários.

Este projeto permitiu uma compreensão profunda das funcionalidades e complexidades envolvidas no uso de tecnologias modernas de comunicação em tempo real, consolidando o conhecimento sobre a aplicação prática de protocolos da camada de aplicação. Além disso, ele também abre portas para a exploração de melhorias e novas funcionalidades, como otimização de qualidade de vídeo, controle avançado de banda e suporte a grandes quantidades de usuários.

A aplicação pode ser acessada em: https://webrtc.puida.xyz.

Caetano Santos Lucio

Foi um aprendizado interessante e psicologicamente desafiador e caótico, que proporcionou diversos conhecimentos que ainda estão se solidificando, mas que proporcionaram uma base interessante não nos conhecimentos sobre WebRTC e WebSockets, aos quais tentarei implementar em outras soluções, mas também conhecimentos que vou levar para a vida. Participei de decisões de implementação e de elaboração do relatório. Eu me dou nota 10 na autoavaliação.

Guilherme Puida Moreira

Eu aprendi bastante sobre WebRTC e WebSockets, e consegui colocar na prática vários conceitos que foram explicados em sala de aula. Eu participei ativamente de todas as etapas do projeto, incluindo a implementação do sistema, elaboração do relatório e gravação do vídeo. Eu me dou nota 10 na autoavaliação.

Irwin Schmitt

Procurei ser proativo no desenvolvimento do projeto, já pensando na arquitetura e na melhor maneira de paralilizar as tarefas assim que o enunciado foi postado. Além do planejamento, também logo implementei a página inicial que lida com a criação das salas. Avalio minha participação como nota 10.

Thiago Vivan Bastos

Aprendi na prática como utilizar os conceitos das camadas de redes, principalmente camada de aplicação. Participei da implementação de código, gravação do vídeo e relatório. Me dou 10 na autoavaliação.

Referências

MOZILLA DEVELOPER NETWORK. *WebRTC: Glossary*. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/WebRTC. Acesso em: 15 set. 2024.

GOOGLE. *WebRTC*. Disponível em: https://webrtc.org/. Acesso em: 15 set. 2024.

WORLD WIDE WEB CONSORTIUM. *WebRTC*. Disponível em: https://www.w3.org/TR/webrtc/. Acesso em: 15 set. 2024.

MOZILLA DEVELOPER NETWORK. *WebSockets API*. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. Acesso em: 15 set. 2024.

WHATWG. *WebSocket Protocol*. Disponível em: https://websockets.spec.whatwg.org/. Acesso em: 15 set. 2024.

ROSENBERG, J.; WEINBERGER, J.; HUITEMA, C.; MAHY, R. *RFC 3489: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. Disponível em: https://datatracker.ietf.org/doc/html/rfc3489. Acesso em: 15 set. 2024.

ROSENBERG, J.; MAHY, R.; MATTHEWS, P.; WING, D. *RFC 5389: Session Traversal Utilities for NAT (STUN)*. Disponível em: https://datatracker.ietf.org/doc/html/rfc5389. Acesso em: 15 set. 2024.

PERKINS, C.; WING, D.; MAHY, R.; PETITHOMME, M. *RFC 5766: Traversal Using Relays around NAT (TURN)*. Disponível em: https://datatracker.ietf.org/doc/rfc5766/. Acesso em: 15 set. 2024.