Salas Imersivas - WebRTC

Danielson Jorge Brito Sanches

Dissertação apresentada

à Escola Superior de Tecnologia e Gestão

para obtenção do Grau de Mestre em Sistemas de Informação.

Trabalho orientado por:

Prof. Nuno Rodrigues

Dr.Rui Ribeiro

Prof. Evandro Alves

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

11-2017

Salas Imersivas - WebRTC

Danielson Jorge Brito Sanches

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação.

Trabalho orientado por:

Prof. Nuno Rodrigues

Dr.Rui Ribeiro

Prof. Evandro Alves

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

11-2017

Dedicatória

Dedico este trabalho à minha família, em especial à minha mãe pelo apoio dado em todas as etapas da minha vida académica, aos meus amigos que também são família, Wizt e MRDQ, aos amigos que a cidade de Bragança me deu e a todos que acreditaram em mim e motivaram na realização deste trabalho.

"Longwinded, running through this life like it was mine, never settling, but setting every goal high, one thousand burpees on the path to my own destruction or success but what is a mistake without the lesson? See, the best teacher in life is your own experience, none of us know who we are until we fail. They say every man is defined by his reaction to any given situation Well who would you want to define you? Someone else or yourself?. Whatever you do, homie, give your heart to it and stay strong."

Nipsey Hussle

Agradecimentos

Agradeço em primeiro lugar a Deus, à família pelo apoio incondicional, aos funcionários do gabinete de serviços informáticos do IPB, em especial ao meu coorientador Evandro Alves pelo suporte diário para o desenvolvimento deste trabalho, ao meu orientador Nuno Rodrigues pela ajuda e esclarecimentos sempre que necessário, ao pessoal da FCCN que esteve encarregue da orientação deste trabalho, nomeadamente os senhores Rui Ribeiro e Paulo Costa, por fim agradecer também ao IPB por disponibilizar os meios necessários para que fosse possível a execução deste trabalho.

Resumo

O projeto salas imersivas foi proposto pela FCCN e tem por objetivo possibilitar a realização de vídeoconferências entre utilizadores desta plataforma com base em tecnologia WebRTC. Trata-se de um projeto já previamente iniciado, sendo objetivo do presente trabalho implementar um conjunto de melhorias, nomeadamente:

- Melhorar a experiência do utilizador na utilização da plataforma;
- Aumentar a segurança, disponibilidade e performance;
- Introduzir novas funcionalidades

Após uma fase inicial de estudo do WebRTC e análise da aplicação, passou-se à fase da instalação da plataforma e componentes para entender em que ponto a mesma se encontrava e, ao mesmo tempo, foi feita uma análise para tentar perceber que caminho seguir para implementar as funcionalidades e melhorias propostas. De seguida, foram implementadas algumas das novas funcionalidades identificadas atrás.

Findo o prazo de desenvolvimento, foram alcançados resultados positivos em relação ao que de novo foi adicionado a este projeto, concluindo assim, ser uma plataforma interessante e uma tecnologia que embora não esteja ainda no seu ponto alto de utilização, já se mostra num bom caminho para ganhar espaço em soluções web de vários tipos.

Palavras-chave: WebRTC, multidomínio, vídeoconferência, fccn.

Abstract

Immersive rooms is a project that belongs to FCCN and its goal is to make possible video-conference between the users in the platform based on a technology called WebRTC. Being a project that was already under development, the goal of this present work is to implement a set of upgrades, namely:

- Make the user experience better while using the platform;
- Improve the security,availability and perfomance;
- Add new features;

After an initial stage of study of the WebRTC and an analysis of the application, was then, made efforts to install the platform and its components to try and understand how it was put together and how it works, at the same tame, thinking of ways to add the new features and make some upgrades. Then were added some of the new features previously identified.

At the end of the development process, were made progress in the context of the new features, where the new features and upgrades added, showed the expected results, showing that is a very interesting platform and a technology altough not in his prime, shows that it have what is needed to make its way to be very useful in a lot of web based solutions for diferent use cases.

Keywords: WebRTC, multidomain, videoconference, fccn

Conteúdo

Lista de Figuras

Siglas

AJAX Assynchronus JavaScript and XML. 32

API Application Programming Interface. 5, 10–12, 32

DNS Domain Name System. 48–50

DTLS Datagram Transport Layer Security. 26

FCCN Fundação para Computação Científica Nacional. 31, 32

HTML Hyper Text Markup Language. 9, 10, 32, 46

HTTP Hypertext Transfer Protocol. 10, 13

ICE Interactive Connectivity Establishment. 19, 21–23, 26, 56

IETF Internet Engineering Task Force. 5, 12

JSEP JavaScript Session Establishment Protocol. 14

JSON JavaScript Object Notation. 27

MCU Multipoint Control Unit. 25

NAT Network Address Translation. 16–21, 23, 26

RTCP Real-Time Transport Control Protocol. 27

RTP Real-time Transport Protocol. 27

SCTP Stream Control Transmission Protocol. 14

SDP Session Description Protocol. 11, 14, 15, 17, 18, 21, 26

SGBD Sistema de Gestão de Base de Dados. 11

SIP Session Initiation Protocol. 12, 27

STUN Session Traversal Utilities for NAT. 17–23, 26

TCP Transmission Control Protocol. 13

TURN Traversal Using Realys aroud NAT. 19–22, 26

URL Uniform Resource Locator. 39

W3C World Wide Web Consortium. 5

WebRTC Web Real-Time Communication. xii, 2, 5, 6, 11, 13, 14, 17, 21, 24–30, 32, 45, 49, 50, 52, 54, 71

XMPP Extensible Messaging and Presence Protocol. 12

Capítulo 1

Introdução

Desde os primórdios da Internet que se percebeu que os casos de uso para esta eram quase incalculáveis, apesar de, nos primeiros tempos, estes mesmos casos de uso serem limitados às tecnologias existentes. Décadas depois, com a evolução que se presencia a cada dia no que concerne à computação e Internet, estes casos de uso se mostram em ainda maior quantidade. Uma área que foi ganhando relevância com o passar dos anos foram as tecnologias relativas à multimédia. A evolução na capacidade dos componentes informáticos e melhorias nas redes de ligação à Internet trouxeram possibilidades ligadas às tecnologias multimédia para que estas assumam papéis que hoje em dia consideramos quase indispensáveis. É comum lembrar-se de um momento que necessitou realizar uma chamada ou vídeochamada para alguém utilizando um software existente no mercado, ou uma ocasião em que necessitou transferir um ficheiro para outra pessoa utilizando alguma ferramenta disponível. Estes são apenas alguns casos que ocorrem e que por vezes se tornam mesmo parte do nosso quotidiano.

O WebRTC, sendo uma tecnologia relativamente recente, trouxe mais uma possibilidade de realizar estas tarefas, para além das ferramentas já existentes no mercado. Este standard dá a possibilidade de implementação dos casos de uso antes mencionados e muitos mais que também são por nós considerados necessários ou até fundamentais. Até aqui nada de novo, mas a maior vantagem que esta tecnologia traz e que a diferencia do resto é que os utilizadores de soluções baseadas em WebRTC não necessitam de instalar qualquer

software ou extensão adicionais, precisando estes de possuir apenas um browser.

Com esta tecnologia é possível criar ferramentas para dar resposta às mais variadas necessidades e, neste projeto, designado Salas Imersivas WebRTC, foi abordada a vertente de vídeoconferência que esta tecnologia permite desenvolver.

1.1 Enquadramento

Este trabalho visa dar continuidade a um projeto iniciado na FCCN, delineado para tirar partido das boas caraterísticas que o standard WebRTC pressupõe. A criação do projeto Salas Imersivas WebRTC visa a criação de uma plataforma de videoconferência, com base nesta tecnologia. O projeto iniciou-se no ano letivo anterior, no serviço técnico de vídeo da FCCN, com o âmbito de um projeto académico, em parceria com uma instituição universitária de Lisboa. Nesta sequência, o responsável pelo referido serviço da FCCN (Dr. Rui Ribeiro), solicitou a colaboração do IPB para dar continuidade ao desenvolvimento do projeto. A proposta inicial visava ultrapassar alguns problemas e limitações que se encontravam na sua versão anterior e ao mesmo tempo procurar desenvolver novas funcionalidades.

1.2 Objetivos

O desafio foi subdividido em três tópicos que representavam os principais aspetos a serem tratados.

- **User Experience** – visa abordar a forma como os utilizadores interagem com a plataforma. É necessário realizar uma análise aos componentes da plataforma e a disposição dos mesmos de forma a perceber se estes estão capacitados para dar resposta a todos os casos de uso que são exigidos pelo utilizador. Também analisando a interação do ponto de vista de usabilidade, onde devemos tentar perceber as dificuldades na utilização da plataforma do ponto de vista do utilizador no que concerne à sua navegação e utilização da plataforma. Questões como quantos cliques

são necessários para ter acesso a um recurso, que novos processos são necessários, mapeamento da plataforma, entre outras questões, são essenciais para ajudar a perceber até que ponto esta se encontra adaptada como um produto de mercado.

- **Segurança, Disponibilidade e Performance** – Sendo uma tecnologia relativamente jovem e dando os seus primeiros passos nos últimos anos, é fácil de perceber que muitas questões relativas à segurança em WebRTC são levantadas. Possíveis falhas de segurança ou falta de robustez são necessariamente prioridade nesta matéria. No que concerne à disponibilidade do sistema, é necessário dotar o mesmo de mecanismos de identificação de falhas e recuperação caso estas aconteçam nos componentes o integram. Relativamente à performance, é imperativo aumentar a escalabilidade da plataforma de forma a garantir acesso a um maior número de utilizadores em simultâneo.

- **Novas Funcionalidades** – Existem algumas funcionalidades que se podiam traduzir em vantagens para a plataforma mas o foco consiste em apenas 3 numa fase inicial. Gravação e reprodução é a primeira pois faz todo o sentido dar aos utilizadores capacidade de gravar as suas chamadas e, quando quiserem, fazer a reprodução das mesmas. A segunda é o *Screen Sharing*, encontrando-se a mesma já quase totalmente implementada, faltando apenas alguns retoques finais para se considerar pronta. Por fim, o Multidomínio, que tem por objetivo homogeneizar o sistema de forma a permitir chamadas entre utilizadores instalados em plataformas diferentes. As salas imersivas serão instaladas em domínios diferentes e é necessário uma forma de fazer com que, para além de ser possível realizar chamadas entre os clientes instalados no mesmo domínio, seja também possível estabelecer conectividade entre clientes em plataformas diferentes, tudo isto de forma transparente.

1.3 Estrutura do Documento

Este documento encontra-se dividido em 6 capítulos, onde cada um se desdobra em sub-capítulos para melhor descrever o conteúdo dos mesmos, com o objetivo de facilitar a interpretação desto mesmo.

O primeiro capítulo é dedicado a uma breve introdução sobre o tema a ser tratado nesta tese. É também feito um enquadramento do trabalho a realizar bem como a especificação dos objetivos iniciais aqui estabelecidos.

No capítulo seguinte é tratado o estado da arte relativo ao WebRTC, onde são caraterizados os pressupostos que compõem esta tecnologia, bem como a arquitetura e tecnologias de suporte a este projeto.

No capítulo três é feita uma descrição detalhada do projeto Salas imersivas WebRTC, descrevendo os seus componentes, a forma como este foi implementado e o ponto onde o mesmo se encontra em termos de desenvolvimento.

No quarto capítulo é apresentado o que de novo foi introduzido neste projeto, em termos de novas funcionalidades e melhorias.

O quinto capítulo é reservado aos testes e às demonstrações de resultados obtidos utilizando o que foi desenvolvido no capitulo anterior.

Por fim, no último capítulo são descritas as conclusões finais e o trabalho futuro que pode ser desenvolvido neste projeto.

Capítulo 2

Estado da Arte

2.1 WebRTC

Web Real-Time Communications (WebRTC) é um standard criado pelo W3C e IETF. Engloba conjunto de protocolos e interfaces de programação que, juntos, formam uma API que visa facilitar a implementação de serviços de comunicação em tempo real através de um simples browser sem necessidade de se instalar qualquer aplicação ou *plugin* adicional. A possibilidade de estabelecimento de canais ponto a ponto entre *browsers* vem abrir possibilidades e casos de uso que outrora se revelariam problemáticos [1].

Com o aparecimento do WebRTC, abriram-se muitos casos de uso que visam tirar proveito da conexão "browser-to-browser" ou "browser-to-non-browser", mas isto acarreta a necessidade de protocolos e normas que vêm sendo definidas para parametrizar este tipo de comunicação que pode ser direcionado a outro browser, equipamento, provedor de serviços de telefonia entre outros casos.

Para entendermos como o WebRTC funciona temos que primeiro perceber como é estabelecida a comunicação utilizando *browsers*. Quando é realizado um pedido num browser através do carregamento de uma página ou de cliques em conteúdos da mesma, este browser faz um pedido a um *web server* através de um protocolo de comunicação, que é transportado até ao *web server* por um protocolo de transporte, e após processar tal

pedido o browser responde da mesma forma, enviando o conteúdo da resposta através de um canal de transporte. O WebRTC não contempla apenas estes casos, pois uma de suas mais valias é a capacidade de estabelecer um canal de comunicação browser-to-browser, onde entram outros fatores a serem considerados, como a sinalização e troca de dados multimédia em tempo real.

2.2 Arquitetura

Sendo o WebRTC capaz de possibilitar muitos casos de uso, é importante saber de que forma implementar o mesmo para que se consiga dar resposta ao que é exigido. É fácil perceber que uma plataforma que implementa partilha de ficheiros exige uma abordagem diferente de uma para *e-learning* ou vídeoconferência, sabendo que estes 3 exemplos são todos possíveis de serem implementados com WebRTC. No fim, o importante é perceber a finalidade que se pretende dar ao WebRTC e o que lhe será exigido dos seus utilizadores finais. Para perceber isso tem-se os seguintes exemplos de modelos de arquiteturas webrtc [2]:

2.2.1 Malha

Consiste no cenário mais simples de perceber, onde não sendo necessário uma infraestrutura complexa, resume-se a conectar cada participante a todos os seus destinatários possíveis com ligações ponto-a-ponto para cada um como está representado na figura 2.1. Em termos de eficiência pode não parecer a melhor solução e, no escopo da gestão de recursos, quer de rede quer de hardware, se apresenta dispendioso, pois abrir uma conexão ponto-a-ponto para cada destinatário num cenário de maior escala exigiria muita largura de banda e capacidade de processamento, mas a verdade é que esta solução garante adaptação de bit rate independente para cada utilizador com o menor atraso possível.

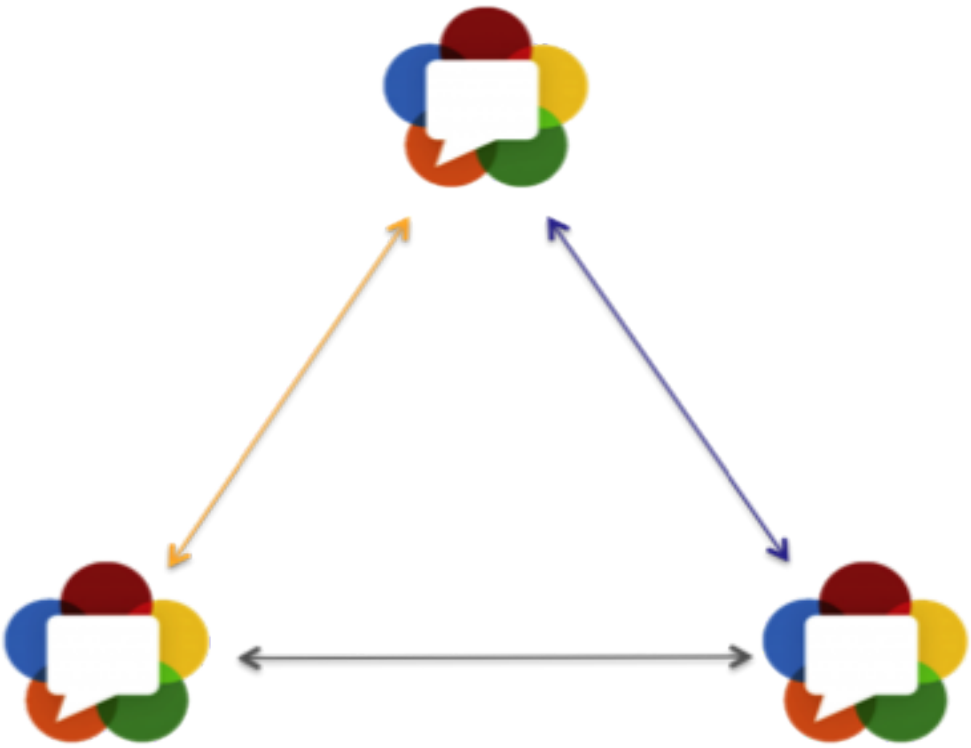

Figura 2.1: Arquitetura em Malha

2.2.2 Mixer

Esta arquitetura é geralmente encontrada nas soluções de vídeoconferência, onde a utilização de um ponto central carateriza esta abordagem. Retirando alguma da necessidade de inteligência nos destinatários, o ponto central (*mixer* da figura 2.2) é responsável pela criação de uma ligação ponto a ponto com cada um dos intervenientes do sistema, tornando-se o centro do mesmo. Também é responsável pela receção e *mixagem* do áudio e vídeo que é transmitido por cada participante e gerar um único *stream* que será reenviado para cada um dos participantes. Esta solução apresenta pontos interessantes como a capacidade de ajuste de bit rate para cada participante, já que o *mixer* pode enviar vídeos com qualidades diferentes para cada recetor. Os maiores entraves encontram-se nos custos para ter uma infraestrutura deste tipo e também como o ponto central ou *mixer* tem que fazer o *decoding* ou *re-encoding*, é fácil chegar à conclusão que haverá atrasos e também possíveis perdas de qualidade [3].

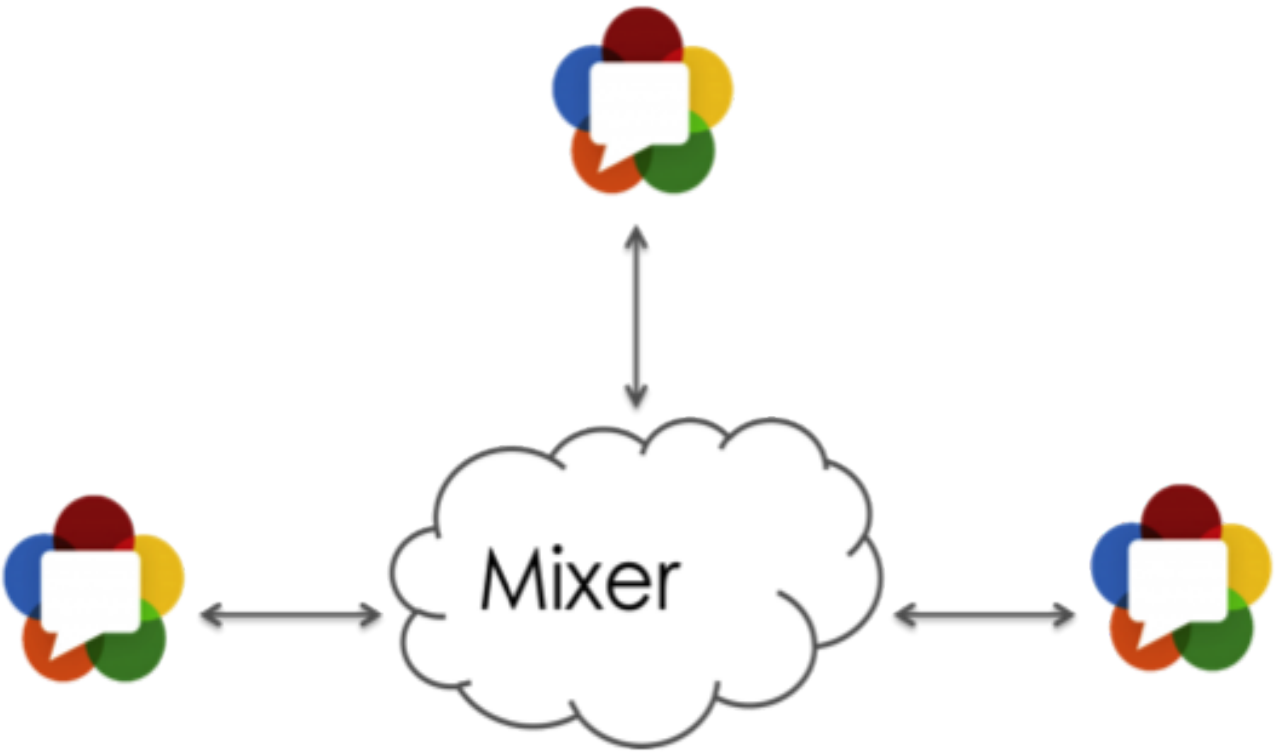

Figura 2.2: Arquitetura Mixer

2.2.3 Router

À semelhança da abordagem anterior, a arquitetura *router* ou *relay* também é construida à volta de um elemento central (figura 2.3), porém este elemento apenas realiza metade do trabalho em relação à arquitetura *mixer*. Nesta abordagem o elemento central é responsável apenas pela inspeção e encaminhamento de pacotes que representam os *streams*, ignorando as funções relativas à codificação, fazendo com que melhore em termos de *delay* e perda de qualidade, pois o *encoding/decoding* fica à responsabilidade do utilizador final [3].

Qual das arquiteturas usar é um pergunta muito subjetiva, pois depende completamente do que se pretende com a implementação desta tecnologia. Existem casos de uso apropriados para cada uma das abordagens e até existem casos onde podem ser todas implementadas em simultâneo. Para soluções de transferência de ficheiros, provavelmente a malha seria a mais adequada, por outro lado, para soluções de vídeochamada levando

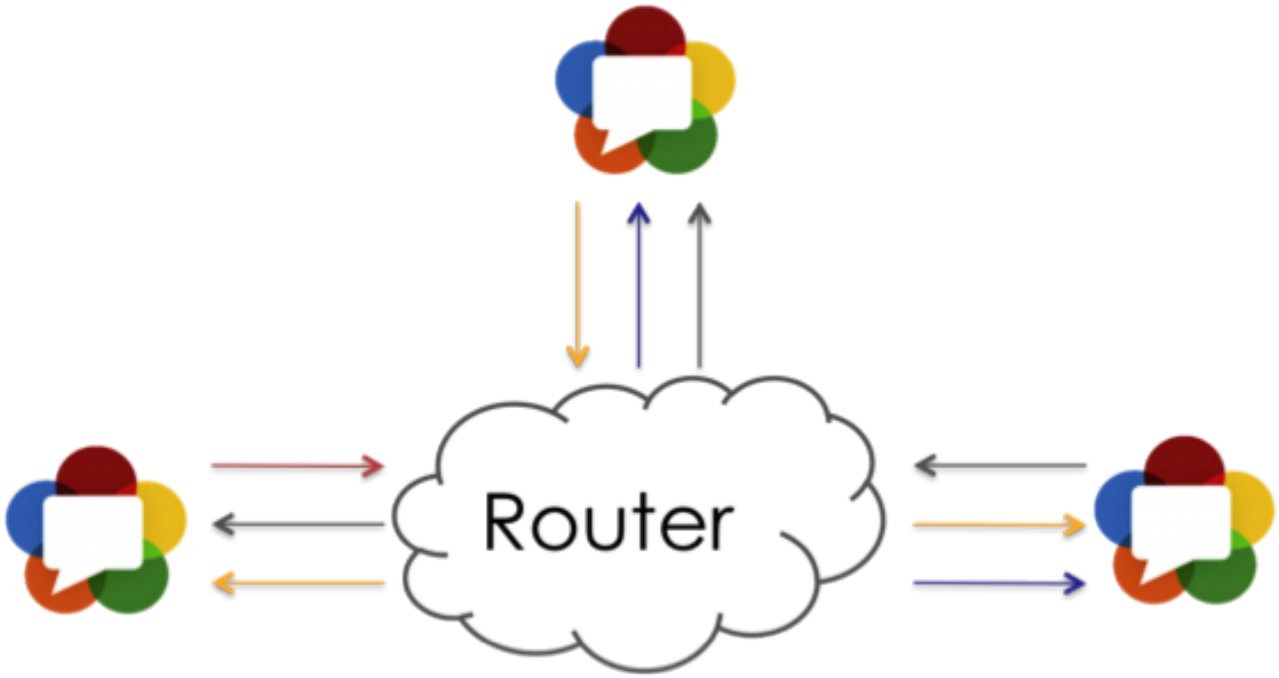

Figura 2.3: Arquitetura Router

em consideração a qualidade que chega aos destinatários da mesma, a solução baseada em mixer seria a mais adequada. Para este projeto em específico a arquitetura que melhor se adequa é a *router* ou *relay*, pelo facto de não se ter conhecimento prévio da quantidade de utilizadores que podem utilizar o sistema. Sendo uma plataforma de vídeoconferência sobretudo, é impossível prever o número de pessoas que podem participar numa chamada fazendo com que a primeira arquitetura seja descartada.

2.3 Tecnologias de suporte ao projeto

- **HTML**
  - Hyper Text Markup Language, é uma linguagem de marcação que serve para criar páginas web que são interpretadas por *browsers*. Na prática, o HTML é usado para criar o esqueleto de uma página para que esta respeite uma estrutura definida pelo criador da página.

- **PHP**
  - Hypertext Preprocessor, é uma linguagem de *scripting* utilizado para dinamizar e tornar páginas web interativas, sendo que pode ser mesclado com HTML tornando-a numa ferramenta poderosa, onde o código é executado no servidor, dando ao cliente apenas o resultado do *script*. Por outras palavras, é uma linguagem de programação *server-side*.
- **JavaScript**
  - É uma linguagem de programação web orientada a objetos, mas ao contrário do php, o javascript é *client-side* e serve para manipular comportamentos no lado do cliente sem ter que interagir com o servidor, podendo alterar conteúdo da página e estabelecer comunicação assíncrona.
- **JQuery**
  - É uma pequena e rápida biblioteca do javascript que serve para simplificar os processos de manipulação, tratamento de eventos, animação, entre outros, através de uma API muito simples de usar.
- **AJAX**
  - Assynchronus JavaScript and XML, não sendo uma linguagem de programação nem uma ferramenta, consiste num conceito de utilização de *scripts client-side* com a capacidade de trocar dados com um servidor e atualizar partes de uma página web, sem a necessidade de recarregar a página por completo utilizando chamadas assíncronas ao servidor.
- **Apache**
  - Consiste num servidor Hypertext Transfer Protocol (HTTP) multiplataforma que serve para alojar *websites* para que estes sejam acedidos.

- **MySQL**

  - Mysql é um sistema de gestão de base de dados baseado em sql que possui grande robustez e elevado desempenho, fazendo com que seja um SGBD de qualidade para dar suporte a plataformas que possuem a necessidade de guardar dados.

2.4 Sinalização

A sinalização é importante para qualquer aplicação que tenha a necessidade de manter uma troca contínua de eventos com uma entidade remota. Embora esteja no modelo implementado e é sempre mencionado em qualquer artigo ou livro que aborde WebRTC, a verdade é que a sinalização não faz parte da API pois não foi incluído no standard. Os responsáveis pelo WebRTC preferiram deixar este escopo na responsabilidade da camada de aplicação, onde cada um que for implementar a API WebRTC terá a flexibilidade de escolher o que melhor se ajusta às funcionalidades que pretende implementar ou simplesmente escolher o que mais lhe apetecer, fazendo com que a escolha do protocolo e serviço a implementar sejam livres [4]. A sinalização em WebRTC pode ser abordada de diversas formas e, como foi antes referido, sempre dependendo de alguns fatores, mas a finalidade da sinalização é sempre a mesma: criar um canal para transportar informação relativa ao estabelecimento da ligação entre clientes como mensagens de controlo, de erro, de configuração de rede e descrição da sessão multimédia.

Embora a escolha do protocolo de sinalização seja livre, existem requisitos que necessitam ser cumpridos neste processo. O WebRTC requer que o servidor saiba interpretar SDP.

SDP ou Session Description Protocol é um protocolo de rede que serve para o que exatamente o nome diz, descrever uma sessão. Consiste num protocolo para descrever a multimédia que será transmitida durante uma sessão. Não negoceia a multimédia em si, apenas serve para trocar informação acerca do tipo de multimédia, formatos suportados

e todas as configurações e parâmetros associados, necessários para o estabelecimento de um canal multimédia [5].

2.4.1 Protocolos de sinalização

Jingle

É um protocolo de sinalização que surgiu da evolução do Extensible Messaging and Presence Protocol (XMPP) onde foi adicionada capacidade de comunicação *peer-to-peer*, desenvolvida pela XMPP Standards Foundation em parceria com a Google. Este protocolo pode ser usado para sinalização com um canal de sinalização XMPP e também para troca de media utilizando DataChannel [6][7].

SIP

Segundo o IETF, o Session Initiation Protocol é um protocolo que serve para que *user agents* se descubram na Internet e caraterizem a sessão que gostariam de partilhar. O SIP fornece a capacidade de enviar e receber registos, convites para sessões e outros tipos de pedidos. Em suma, o SIP pode ser visto como uma ferramenta genérica para criar, alterar ou terminar sessões independentemente do protocolo de transporte ou do tipo de sessão que está estabelecida [7][8].

PEERJS

Para a plataforma salas imersivas, no escopo da sinalização não foi escolhido nenhum dos protocolos acima descritos, mas sim PeerJS. **PeerJS** é um API intermediário implementado em JavaScript que, utilizando NodeJS, serve para gerir conexões de clientes peerjs que representam os extremos de cada lado de uma ligação. É, assim, possível dar suporte de sinalização a aplicações de alta escalabilidade. PeerJS possui duas facetas, uma que representa o servidor, onde os peers são criados, destruídos, ou controlados através da troca de mensagens. Existe também o lado cliente, onde cada cliente pode pedir a criação de

um peer, que é atribuido apenas um ID e também pode estabelecer a troca de mensagens com outros peers do servidor utilizando o ID que lhe foi atribuído quando criado [9].

2.4.2 Canais de Comunicação

O WebRTC requer um canal de sinalização bidirecional entre browsers. Na definição WebRTC são abordados três: o conhecido HTTP, WebSockets e DataChannels.

WebSockets

WebSocket é um protocolo de rede que permite comunicação entre *browsers* por duas vias entre um cliente local e um host que se encontra em um ambiente remoto na Internet. Quando se abordava a criação de aplicações web entre um cliente e um servidor era necessária a criação de muitas conexões HTTP ao servidor para manter a sessão atualizada enquanto se enviava outras chamadas HTTP ao servidor com notificações. Esta situação resultava em problemas, pois o servidor nessas condições era forçado a criar conexões TCP para cada cliente, uma para enviar informação ao cliente e uma nova para cada mensagem a receber, sendo que do lado do cliente também trazia desvantagens pois o código do lado deste era forçado a manter um mapeamento de todas as conexões para fora e das que recebia para saber identificar as respostas aos pedidos enviados. Surgiu com o objetivo de suprimir a necessidade de envio em massa de pedidos HTTP, quando este é utilizado na camada de transporte. WebSockets assenta sob alguns objetivos e infraestruturas do HTTP porém colmata o objetivo pretendido criando uma única conexão TCP para passar tráfego bidirecional. Uma conexão começa com um pedido HTTP, que logo é atualizado para WebSocket. Para que um servidor WebSocket esteja funcional é necessário que este possua um endereço IP público e ter um servidor HTTP [7][8][10].

DataChannel

Quando se fala das especificações do WebRTC e seus componentes, existe sempre a tendência de descrever tudo a partir de um ponto de vista relativo à multimédia, mas a

verdade é que o WebRTC pode especificar standards tanto para a troca de media, como para transporte de não media. O WebRTC especifica troca de dados que permita a implementação de outros tipos de serviços como chats, partilha de ficheiros, *gaming* entre outras implementações possíveis. A implementação desses serviços por vezes torna-se complicada pois existe sempre a preocupação com fatores como latência ou privacidade. DataChannels surge como uma solução baseada em Stream Control Transmission Protocol (SCTP) que alivia estes problemas pois remove a necessidade de existência de um servidor para fazer o redirecionamento dos pacotes, sendo que os dados são transmitidos numa ligação ponto-a-ponto entre os envolvidos, fazendo com que existam menos saltos na entrega de pacotes e consequente redução na latência ao mesmo tempo que oferece um canal seguro para transmissão de dados de um modo genérico [7][11].

2.4.3 Sessão

Em todo o processo de sinalização é mencionado o estabelecimento ou criação de sessão. Para melhor se perceber o que é e como são criadas, a figura 2.4 é um exemplo baseado na especificação JSEP (JavaScript Session Establishment Protocol) [7].

JSEP

JSEP ou JavaScript Session Establishment Protocol não sendo um protocolo, é mais facilmente interpretado como um modelo que providencia um meio para manipular estados de sinalização ou negociação em aplicações web, permitindo que protocolos de sinalização como os anteriormente descritos sejam implementados em cima deste [7][12].

Exemplo de uma sessão baseada em JSEP [13]:

Assumindo que dispomos de dois utilizadores, **X** e **Y**.

1. X cria uma oferta contendo o seu SDP local

2. X anexa a sua oferta num objeto RTCPeerConnection

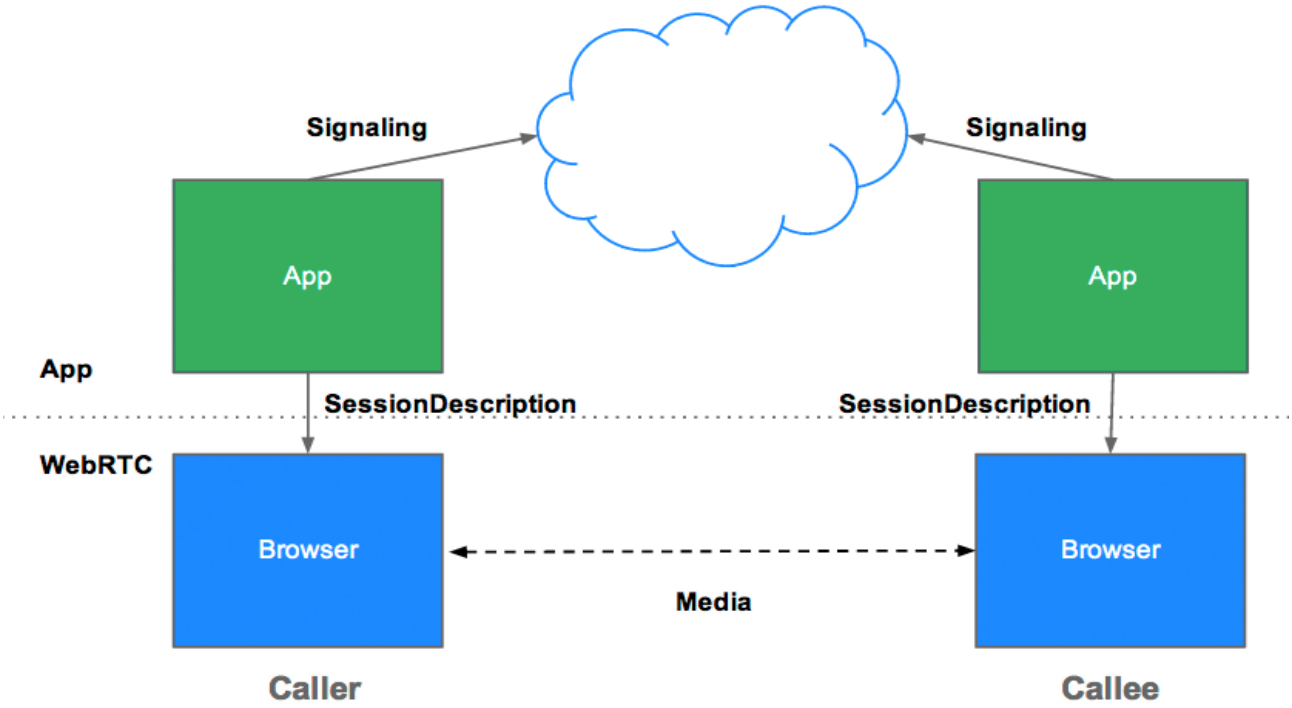

Figura 2.4: Arquitetura JSEP

3. X envia a sua oferta para o servidor de sinalização utilizando WebSockets

4. Y recebe a oferta de X via WebSocket

5. Y cria a resposta com o seu SDP local

6. Y anexa a sua resposta, juntamente com a oferta de X no seu objeto RTCPeerConnection

7. Y envia a sua resposta ao servidor de sinalização utilizando WebSockets

8. X por fim recebe a resposta de Y via WebSocket

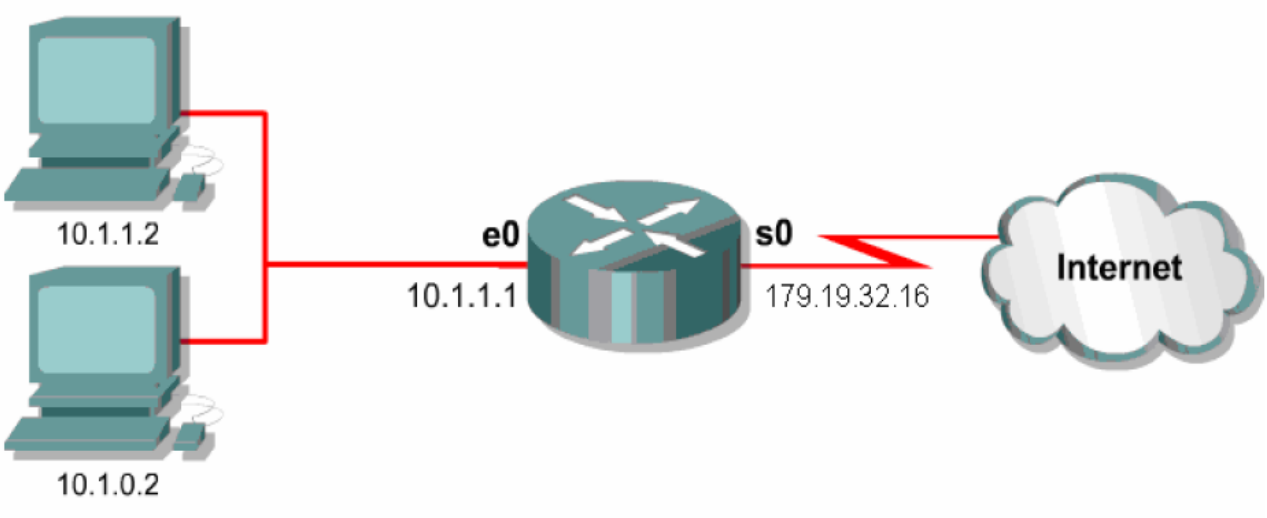

Figura 2.5: NAT

2.5 Descoberta

NAT

Um dos problemas que surgem naturalmente quando se fala de conexões a alvos que estejam numa rede que seja diferente da de origem, é o facto de muitas corporações utilizarem NAT. Network Address Translation, ou NAT, é um standard que foi criado para ajudar a combater o esgotamento de endereços IP disponíveis, causado pelo constante crescimento da Internet e de dispositivos conectados. Na prática, é um método de re-mapeamento de uma rede privada através de um endereço público (figura 2.5). Para além da economia de endereços IP públicos, o NAT também introduziu uma vertente de melhoria na segurança de uma rede, pois ao mascarar os endereços IP dos seus recursos privados com um único endereço público faz com que se consiga limitar o conhecimento/acesso aos recursos de uma rede, limitando a comunicação apenas às conexões que foram requisitadas ou permitidas por dispositivos dentro da rede privada, desde que estas conexões sejam respostas, tornando o *router* numa espécie de *firewall* adjacente [14][15].

Tipos de NAT

- **Static** – Consiste no mapeamento direto de um endereço interno da rede privada para um endereço registado público, mapeamento feito pelo *router* [16].
- **Dynamic** – Tem a função de mapear um endereço interno da rede privada para um endereço disponível registado dentro de um grupo de endereços públicos, sendo que será atribuído o primeiro endereço disponível [16].
- **PAT ou Overloading** – Este caso de NAT acontece quando vários endereços dentro da rede privada são mapeados para o mesmo endereço externo registado. O mapeamento de cada computador dentro da rede é feito utilizando não apenas os respetivos IP's mas também incluindo o porto no seguinte formato x.x.x.x:y , onde x.x.x.x é um endereço IP e y o porto [16].

É fácil notar que, para além das vantagens mencionadas acima, o NAT também introduz dificuldades, sobretudo quando se fala de descoberta de *hosts* ou serviços em redes que se encontram atrás de NAT, para além da questão da perda de performance, tornando aplicações mais lentas. Mas contemplando sobretudo na descoberta de serviços em redes escondidas atrás de NAT, o WebRTC propõe a utilização de dois standards que visam ultrapassar este problema, utilizando STUN e TURN.

2.5.1 STUN

Session Traversal Utilities for NAT é um protocolo de rede que serve para contornar o problema criado pela existência de NAT. Sem uma forma de traduzir os endereços privados em seus respetivos correspondentes endereços públicos, seria difícil aplicações estabelecerem conexões para troca de dados. O STUN permite a um host que pretende comunicar para fora de sua rede privada saber o seu endereço IP e porto públicos que constam na tabela de endereços gerida pelo *router* que tem NAT implementado. Essa informação será utilizada para comunicar com o host remoto no envio do pacote SDP a quando do estabelecimento da sessão.

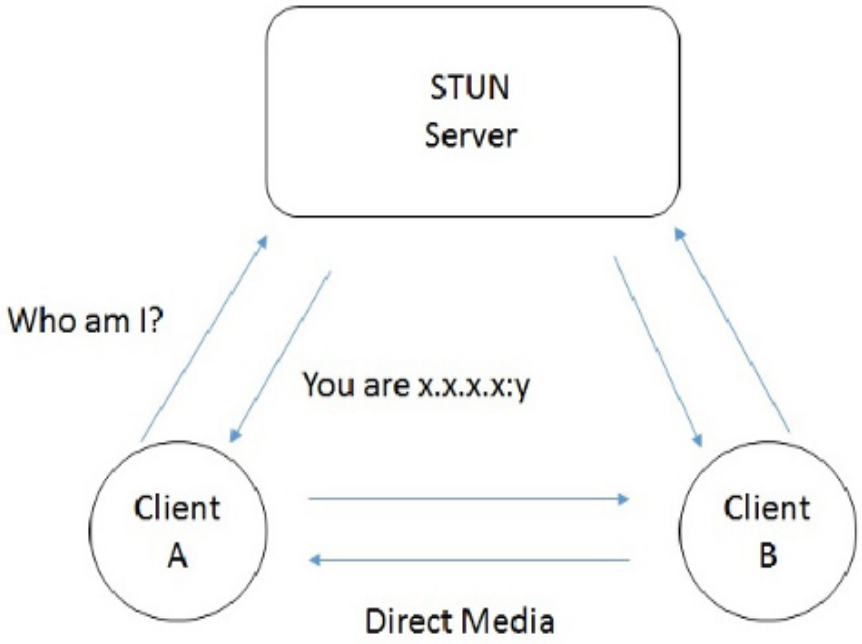

Figura 2.6: Como o STUN funciona

Na prática, para obter esta informação, o cliente STUN que quer saber seu endereço e porto, antes de enviar informação relativa ao SDP, faz uma verificação no seu servidor STUN para saber se se encontra numa rede mascarada por NAT como demonstra a figura 2.6.

O STUN, sendo um protocolo do tipo cliente servidor, funciona através de pedidos. Dois tipos de pedidos são implementados neste protocolo: *request/response* e *indication*. Nos pedidos *request/response*, o cliente que origina o pedido fica à espera de uma resposta por parte do servidor que recebeu o pedido. No núcleo do protocolo apenas existe um método chamado **Binding**. Este método é utilizado pelo cliente STUN para criar e descobrir mapeamentos NAT. Presumindo um cliente e um servidor NAT, onde ambos estão em redes distintas, quando o cliente envia um *request* com *binding*, fará com que o NAT crie uma entrada no mapeamento e atribua um IP e porto. Isto fará com que seja criada uma regra no filtro acerca de quem tem permissão para utilizar este mapeamento

para mandar pacotes para dentro da rede privada. Quando o servidor STUN recebe o pedido *binding*, guarda o IP e porto que recebeu no pedido. Esta informação é reenviada para o cliente STUN e este usa este endereço IP e porto para comparar com o endereço e porto que enviou no primeiro *binding*. Caso sejam diferentes, significa que se encontra atrás de pelo menos um NAT. Na figura 2.7 temos o exemplo de um pacote STUN que é utilizado para troca de mensagens. Um *binding* enviado como indicação serve para impedir o *timeout* de mapeamentos quando são enviados em intervalos regulares de tempo. Basta que o intervalo de envio deste tipo de pedido seja menor que o temporizador NAT UDP mais curto entre cliente, para que o mapeamento não seja descartado [7][12][17].

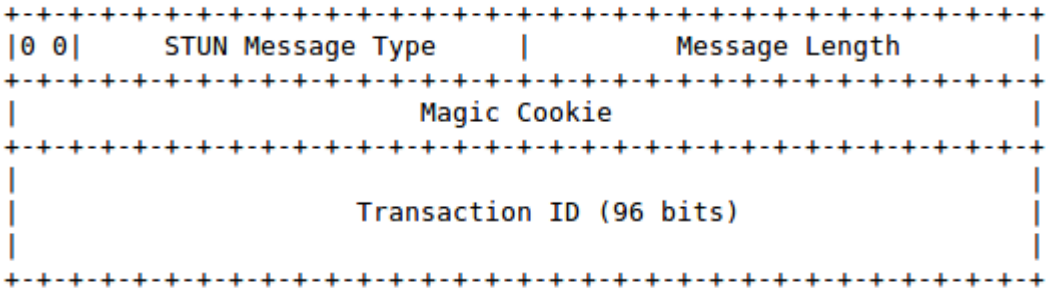

Figura 2.7: Cabeçalho STUN

Em termos de autenticação existem dois mecanismos possíveis em STUN: Short term mechanism e long term mechanism. Short term mechanism é implementado utilizando um *username* e uma password que são válidos apenas durante uma sessão. É utilizado pelo Interactive Connectivity Establishment (ICE) para verificar se cada conexão é única. Long term mechanism utiliza um mecanismo de desafio e resposta [17].

2.5.2 TURN

Traversal Using Realys aroud NAT ou TURN funciona como uma extensão do protocolo STUN. TURN permite a clientes comunicar com um servidor TURN e pedir para alocar um endereço IP e porto na sua instância, para que depois possa retransmitir pacotes entre o servidor e o cliente em si (figura 2.8). De um modo genérico, podemos

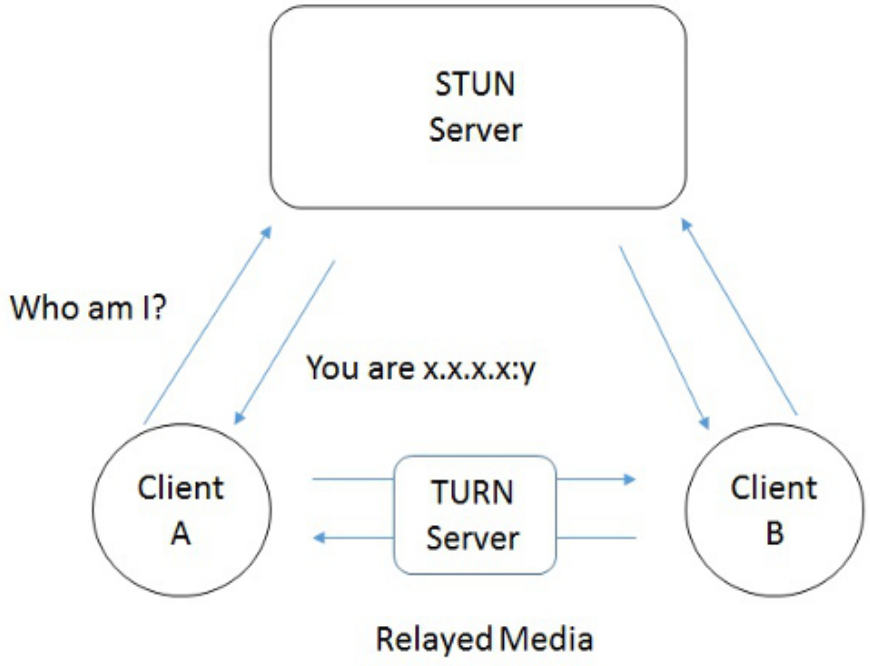

Figura 2.8: TURN

ver o TURN como um retransmissor. O STUN traz imensas vantagens porém não cobre todos os cenários possíveis, já que em casos de NAT que sejam configurados com "address-dependent mapping" ou "address and port-dependent mapping", as técnicas de "hole punching"[7] ou perfuração falham fazendo com que não se estabeleça um canal de comunicação direto entre os dois pontos finais [18].

Criada a necessidade de existir um elemento que sirva de intermediário para o estabelecimento da comunicação entre dois pontos, o TURN, que geralmente se encontra numa rede pública, atua como moderador da comunicação, onde o cliente TURN pede um canal como descrito anteriormente ao servidor TURN para que lhe possa enviar pacotes e o servidor TURN retransmita estes pacotes ao peer de destino. Um cliente utilizando TURN é obrigado a possuir um meio de comunicar o endereço de encaminhamento de pacotes aos seus peers e também de saber o endereço dos seus peers. Uma forma de ultrapassar este entrave é utilizar um protocolo de *rendezvouz* para trocar a informação necessária.

Nestes casos, o TURN é utilizado juntamente com o ICE [7][12][19].

2.5.3 ICE

Interactive Connectivity Establishment ou ICE é um protocolo de rede implementado sob o modelo de troca de SDP em duas fases, para o estabelecimento de sessões multimédia baseado em oferta/resposta [7]. ICE possui duas importantes funções relacionadas com o WebRTC, tornando-a fundamental para a implementação deste [20]:

- ICE permite a clientes que implementem WebRTC fazer a troca de conteúdo multimédia entre dispositivos que estejam em redes com NAT.
- Providencia mecanismos de verificação de consentimento da comunicação. Esta faceta de segurança permite que apenas sejam enviados pacotes a *browsers* que esperam receber tais pacotes.

O ICE tira partido do STUN e TURN para o estabelecimento de sessões numa sequência de cinco fases [7]:

1. Reunir endereços de transporte de candidatos, que consistem num endereço IP e o número do porto, onde possivelmente será possível receber multimédia através de um Peer Connection. Estes endereços são reunidos no momento da chamada, em que este endereço pode representar um dos possíveis quatro tipos de candidatos:
    - Host – é um endereço dado pelo sistema operativo que representa um interface da placa de rede. Se se encontrar atrás de um NAT, será um endereço privado.
    - Server Reflexive – endereço de transporte obtido através de um pedido do cliente ao servidor STUN. Se estiver atrás de um NAT, então será o seu endereço público.
    - Peer Reflexive – endereço de transporte também dado pelo STUN, porém através do outro peer (ICE agent). É um novo candidato que é descoberto durante a verificação de conectividade.

- Relayed – endereço de um retransmissor de media obtido quando é feito um pedido de alocação a um servidor TURN.

2. Fazer o câmbio de candidatos pelo canal de sinalização. Antes de enviar os candidatos, estes são ordenados e priorizados pela ordem dos tipos de candidatos como mencionados no ponto anterior. Se houver preferência entre IPv4 e IPv6, este tipo de priorização é feito de outra forma.

3. Realizar os testes de conectividade STUN assim que tenha enviado e recebido os candidatos. Nesta fase, são geradas as respostas para todos os pedidos de conectividade dos peers que tenham passado a fase de autenticação e serão colocados em fila na seguinte máquina de estados:

   - Frozen – estado de espera enquanto aguarda que as verificações estejam prontas a serem realizadas;
   - Waiting – quando o algoritmo de verificação de conectividade ICE dita que estas estão prontas a serem realizadas, passa para este estado de espera com o intuito de gerir os testes realizados para não serem enviados todos em simultâneo;
   - In-Progress – quando recebem permissão para começar os testes e é enviado a verificação para o outro peer;
   - Se receber uma resposta do peer, o estado é alterado para *succeeded*. Caso a verificação sofra um *timeout* sem a resposta, então o estado é alterado para *failed*;

4. Escolher o par de *endpoints* e começar a troca multimédia. As verificações continuam até que todas cheguem a um dos dois estados finais, ou um par tenha sido escolhido. A escolha de pares é feita pelo ICE Agent que controla a sessão. Este protocolo (ICE) possui um algoritmo para escolher qual browser é o ICE Agent controlador e o ICE Agent controlado. O ICE Agent controlado é informado que um par foi

escolhido quando recebe uma verificação de conectividade STUN por parte do ICE Agent controlador. Então, o controlado responde indicando que este par escolhido será utilizado para trocar media, ficando os dois *browsers* prontos a trocar media entre si.

5. Enviar *keepalives* para manter a sessão ativa pois, para garantir que o mapeamento NAT e os filtros não expirem durante uma sessão, é necessário o envio de testes de conectividade num intervalo de quinze segundos, mesmo que pacotes de media não estejam a ser transmitidos, fazendo com que a sessão esteja sempre ativa e pronta a receber ou enviar media.

Caso seja detetada alguma mudança de endereço, o IP realiza um ICE restart, que basicamente consiste em saltar para o primeiro passo e reunir outra vez os candidatos. Basta um simples recarregar da página que contenha o PeerConnection para disparar o ICE Restart.

2.5.4 Trickle ICE

O ICE consiste na aquisição, agrupamento, organização e disposição por par dos candidatos para que possam ser feitos os testes de conectividade e consequente escolha do par para dar inicio à troca de media. É fácil perceber que embora este processo sequencial seja vantajoso no ponto de vista de quem vai implementar esta topologia, na prática, sujeita o sistema a perdas de tempo pois tem que passar todos estes estados e fases nos dois extremos da chamada antes de estabelecer conexão entre estes. Com isso surgiu a evolução do ICE, um protocolo denominado Trickle ICE, que apareceu há alguns anos com o intuito de simplesmente melhorar o processo de estabelecimento de uma chamada. Mais concretamente, o Trickle ICE implementa a mesma arquitetura que o ICE, porém com uma grande diferença: os mesmos processos do anterior são executados simultaneamente, e isto faz toda a diferença de eficiência e tempo de estabelecimento de uma chamada. Enquanto o ICE tenta reunir todos os candidatos que encontrar antes de os enviar para o alvo da chamada e começar a negociação, o Trickle ICE vai enviando os candidatos

à medida que os descobre fazendo com que seja possível ir verificando se os candidatos descobertos estão prontos a serem utilizados. Enquanto o processo de descoberta e envio acontece, também em simultâneo são realizados os testes de conectividade para que quando se pretender mesmo fazer a chamada, estejam todos os envolvidos prontos para receber a mesma, fazendo com que seja possível estabelecer uma chamada em poucos de milissegundos [7][21].

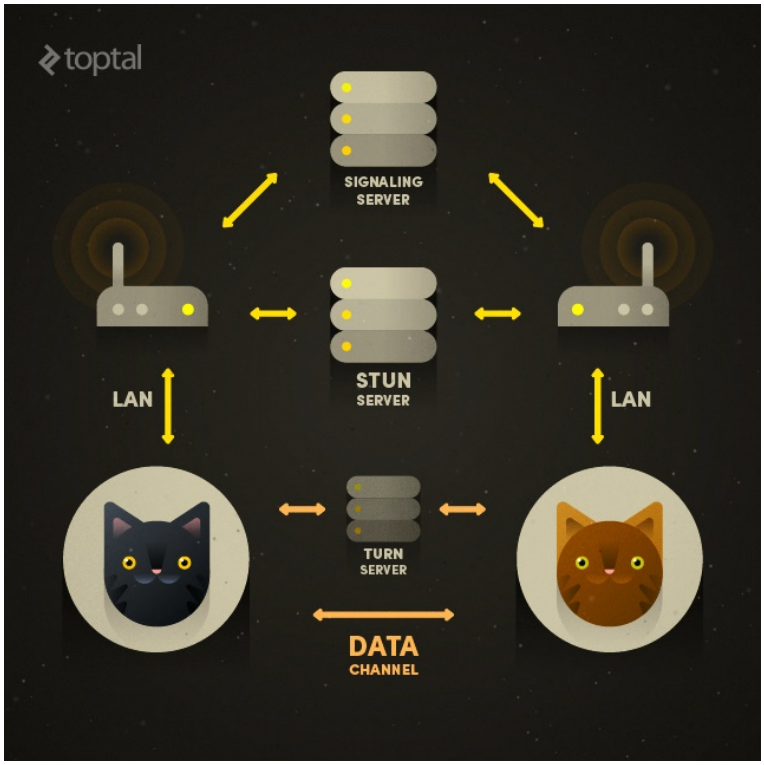

Figura 2.9: Modelo completo da sinalização
10

A figura 2.9 representa graficamente todo o processo de sinalização usado numa sessão WebRTC

2.6 WebRTC Gateway

A palavra *gateway* parece estar um pouco fora de contexto visto que a primordial ideia do WebRTC é estabelecer conexão ponto-a-ponto entre dois extremos de uma chamada sem a

necessidade de nenhum agente pelo meio. Isto não deixa de ser verdade, porém o aumento dos casos de uso que esta tecnologia ganhou, obrigou a que fossem abordadas novas funcionalidades e caraterísticas que fariam sentido implementar na camada de aplicação.

2.6.1 Media Gateway

Quando se fala de ligação ponto-a-ponto temos que ter em atenção que não podemos considerar apenas que os dois pontos sejam *browsers*. Num exemplo simples de perceber, o outro lado da conexão pode ser, não um *browser*, mas sim uma aplicação. Aplicação esta que por sua vez pode estar a funcionar como uma Multipoint Control Unit (MCU), ou media *recorder* ou simplesmente esteja implementado para servir de ponto de transação para uma outra tecnologia [22] como está representado na figura 2.10.

Esta aplicação que, necessariamente tem que implementar a maioria ou totalidade dos protocolos utilizados em WebRTC para poder entregar do outro lado, é o que são chamados WebRTC Gateways, onde um lado tem que obrigatoriamente interpretar WebRTC, enquanto o outro fica à mercê de quem implementa a aplicação e os serviços que pretende disponibilizar através da mesma.

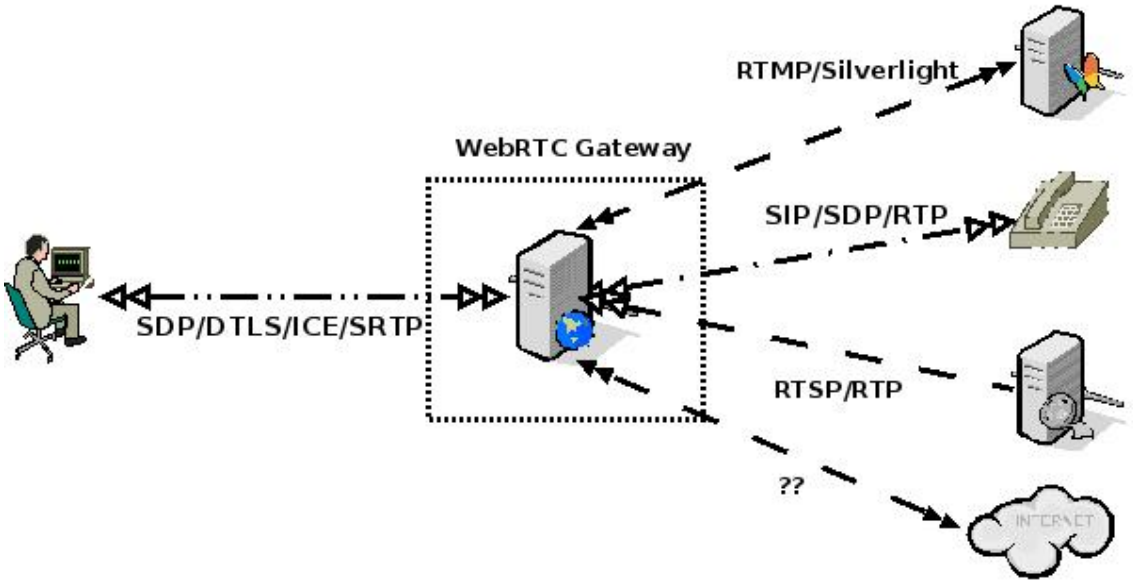

Figura 2.10: Modelo de comunicação de um gateway WebRTC

Existem soluções para praticamente todos os casos que possam ser pretendidos em aplicações que sigam esta arquitetura. Casos como o **Asterisk**, **Kamilio**, **Twilio**, **Janus**, **Kurento** ou **Jitsi** são provavelmente os mais comummente conhecidos nesta secção do WebRTC, com especial ênfase para o Janus Gateway, que foi o eleito para ser utilizado neste projeto. Mas para escolher uma solução das que já foram desenvolvidas ou até pensar em criar uma de raiz é necessário ter em conta alguns pontos que ditam a implementação do mesmo.

Arquitetura

O primeiro ponto é a escolha da arquitetura que melhor se encaixa no que se pretende implementar. Uma arquitetura mais centralizada e homogénea tem como caraterística identificadora a implementação e gestão da sinalização e do tratamento multimédia serem feitos em conjunto, o que a torna esta abordagem mais fácil de ser visualizada. A segunda opção seria implementar os componentes mencionados acima separadamente, decompondo a arquitetura em dois planos separados e criando um meio para fazer estas mesmas interagirem entre si. Tal permitiria uma gestão separada tornando possível abordar questões como a escalabilidade de uma forma mais focada [22].

Protocolos

Sendo o WebRTC uma tecnologia nova e ainda numa fase em que os standards continuam a ser definidos, e devido ao elevado número de casos de uso, torna-a um pouco vulnerável a problemas relacionados com compatibilidades. Existem imensas bibliotecas disponíveis para auxiliar na implementação do que quer que seja no contexto WebRTC, seja relacionado a TURN/STUN/ICE, DTLS, SDP, NAT's heterogéneos, etc... e cada um destes pode estar implementado numa linguagem diferentes. Não é difícil chegar à raiz do problema, tornando-se evidente que criar um *gateway* é sinónimo de implementar uma aplicação que consiga lidar com todos estes protocolos e suas implementações [22].

Media

Neste ponto é onde se deve pensar ou repensar a forma como serão tratados os inputs ou outputs que passarão pelo *gateway* a implementar, assumindo que o mesmo poderá ser utilizado de diversas formas, como um *media recorder*, como *media relay*, como agente de tradução de RTP para outro protocolo ou formato, ou então de conteúdo de uma fonte externa e que precisa ser enviado para um alvo que implemente WebRTC [22].

Sinalização

Embora muitas implementações estejam viradas para o SIP, a verdade é que o standard WebRTC deixa esta faceta ao encargo de quem vá aventurar-se na sua implementação, e isso também inclui implementação de *gateways*. A verdade é que também como as outras implementações WebRTC, a sinalização nos *gateways* também depende da finalidade que esta irá ter [22].

2.6.2 Janus

Segundo os seus criadores (Meetecho), Janus consiste num WebRTC *gateway* com um propósito geral que por isso não oferece quaisquer outras funcionalidades a não ser implementar os meios para estabelecer uma comunicação multimédia WebRTC com um browser através da troca de mensagens JSON e servir de retransmissor de RTP/RTCP e mensagens entre *browsers* e a aplicação *server-side*. Quando se utiliza esta solução, qualquer funcionalidade tem que ser implementada *server-side* através da utilização de *plugins* que os *browsers* possam aceder pelo *gateway* e tirar vantagem de suas funcionalidades. A justificação por trás desta implementação baseada em *plugins* provém da ideia de permitir que quem vá fazer uso deste *gateway* tenha liberdade para utilizar apenas o que necessita para dar resposta às necessidades da sua aplicação. Sendo assim, tem a possibilidade de tanto implementar um *gateway* que abranja todos os casos de uso da arquitetura WebRTC num sistema robusto, como também pode disponibilizar apenas algumas funcionalidades,

consoante o que a aplicação *server-side* requerer. Através dos *plugins* disponíveis, é possível identificar os casos de uso e as possibilidades que o Janus Gateway possibilita a quem pretende utilizar esta ferramenta [22][23].

Plugins disponíveis

- Audio Bridge
- EchoTest
- Record & Play
- SIP
- Streaming
- TextRoom
- Video Call
- Video Room
- Voice Mail

2.7 Soluções baseadas em WebRTC

Atualmente, o WebRTC encontra-se bem distribuído no mercado através de várias soluções para os mais diversificados casos de uso, alguns mais comuns que outros. Soluções de vídeoconferência são os mais populares de se encontrar baseados em WebRTC. Mas também não podemos esquecer das outras áreas que já estão sendo abrangidas por esta tecnologia, áreas como vídeovigilância, *gaming*, prestação de *cloud services*, *live streaming*, telefonia entre outros mais [24][25][26].

Lista-se, de seguida um conjunto de soluções baseadas nesta tecnologia:

- **Helleo** é uma solução de vídeoconferência simples, dispensando informação acerca do estado do contacto, dispensando tempo de espera ou toque de chamada, apenas sendo necessário ligar-se ao contacto se o mesmo se encontra presente.

- **Sococo** por sua vez é uma implementação WebRTC que visa simular um local de trabalho sem que os participantes estejam necessariamente a partilhar a mesma localização, oferecendo os meios para que tal aconteça e chegando mesmo a utilizar um mapa virtual para representar os membros da equipa numa sala comum de trabalho.

- **ShareDrop** é uma plataforma que tem o simples propósito da partilha de ficheiros, tanto entre dispositivos na mesma rede como em dispositivos em redes diferentes, apenas com um "drag-and-drop".

- **Dialoga** sendo uma empresa ligada à telefonia, fornece diversos serviços como PBX ou ACD, tudo baseado em WebRTC.

- **Temasys** é uma solução de *cloud services* que oferece qualquer funcionalidade possível em WebRTC, deixando apenas para o cliente decidir que serviço pretende disponibilizar na sua plataforma ou aplicação. **Clonus** é um outro exemplo de *cloud service provider* para WebRTC

- Um caso de uso muito interessante para o WebRTC seria a monitorização por vídeo e o **Camio** fornece esta funcionalidade, dando a possibilidade de ter diversas câmaras convergindo para um ponto de controlo. **FlashPhoner** também oferece o mesmo serviço

- Um outro conceito muito invulgar e que, se bem analisado é até muito interessante e faz todo o sentido: o conceito de *stream sharing*. Um exemplo disso é o que o **Rabbit** nos traz, uma plataforma onde é possível ter salas com *streams* de vídeos, sejam estes de que tipo for, sendo vistos em simultâneo por todos os participantes desta mesma sala, e com um chat lateral onde podem comunicar entre si em tempo real.

O **Airtime** vai mais além e adiciona a capacidade de estabelecer vídeoconferência entre participantes, enquanto estes assistem a um vídeo juntos (na mesma sala) ou ouvem música.

- Imagine ter um projeto em PHP se encontra encurralado por erros que não consegue perceber as suas causas. Seria muito mais fácil ter ajuda local e tentar resolver o problema. O **Codementor** traz esta funcionalidade, sendo possível, através dele, estabelecer uma chamada vídeo com *experts* em matéria de programação e ao mesmo tempo fazer *screen share* do seu código, para que este lhe ajude a resolver o problema.

- **LiveNinja** oferece uma solução para fazer aproximar empresas e seus seus clientes. Muitos clientes tendem a querer simplesmente trocar mensagens com empresas a quem tenham contratado algum serviço ou adquirido algum produto, mas a maior parte das empresas não fornece este serviço. Eis que o LiveNinja entra em ação com uma aplicação para troca de mensagens entre os dois agentes.

- E se fosse possível realizar *castings* de voz utilizando WebRTC? Ao que parece é mesmo um serviço existente no **Bodalgo**, onde é possível fazer audições de voz e atribuir trabalhos a quem a voz melhor se adeque.

- Existe também uma empresa chamada **Bridge Patient Portal** que fornece serviços relacionados com a área de saúde, onde através da sua plataforma ou aplicação, ajuda empresas do ramo a manterem informação dos seus pacientes atualizada com vista a ajudar as mesmas a melhorarem os seus serviços e relações com os seus pacientes.

Capítulo 3

Plataforma Salas imersivas

3.1 A FCCN e o projeto Salas imersivas

A Fundação para Computação Científica Nacional (FCCN) foi criada em janeiro de 1987 tendo como fundadores a Junta Nacional de Investigação Científica e Tecnológica (JNICT), o Laboratório Nacional de Engenharia Civil (LNEC), o Instituto Nacional de Investigação Científica (INIC) e o Conselho de Reitores das universidades portuguesas (CRUP).

Atualmente, FCCN é uma unidade da FCT – Fundação para a Ciência e a Tecnologia que tem como missão principal o planeamento e gestão da RCTS – Rede Ciência, Tecnologia e Sociedade. A RCTS é uma infraestrutura de investigação digital, transversal a todas as áreas do conhecimento e cobrindo todo o território nacional. Gerida e operada pela FCCN, a unidade da Fundação para a Ciência e a Tecnologia (FCT) responsável pela computação científica nacional, a RCTS oferece aos investigadores, professores e alunos uma infraestrutura digital de alto desempenho que apoia os projetos que desenvolvem a nível nacional e internacional [27]. Os seus serviços são disponibilizados através de uma rede de alto desempenho para instituições de ensino e investigação, assegurando assim os requisitos de comunicações e serviços digitais avançados das diversas comunidades de utilizadores destas entidades. A RCTS constitui-se igualmente como uma plataforma de

experimentação para aplicações e serviços avançados de comunicações [27].

A FCCN disponibiliza uma vasta gama de serviços às comunidades de investigação e instituições de ensino superior que têm-se provado fulcral no desenvolvimento da ciência em Portugal, serviços nas seguintes áreas:

- Computação
- Conectividade
- Colaboração
- Conhecimento
- Segurança

Com vista na promoção e melhoria na capacidade de colaboração dos membros da comunidade de investigação e ensino nacional, a FCCN disponibiliza diversos serviços que ajudam nesta tarefa. O projeto Salas Imersivas WebRTC surgiu neste mesmo âmbito visando conseguir um produto final que sirva para melhorar a tal capacidade de colaboração, assim como tem feito outros serviços já existentes desenvolvidos pela FCCN, como o caso do **Colibri**, **VideoCast**, **Estúdio** e **FileSender** [27].

3.2 Salas Imersivas

A figura 3.1 representa a arquitetura geral do sistema Salas Imersivas WebRTC.

3.2.1 Web Server

Representa a plataforma em si, onde interagimos com o servidor através do browser. É neste servidor que está instalada uma instância da plataforma das salas imersivas, criada com base na API WebRTC e utilizando HTML, php, javascprit, jquery e AJAX. Para manter a plataforma online foi utilizado o servidor HTTP *open source* apache2 e para a gestão de base de dados o motor mysql.

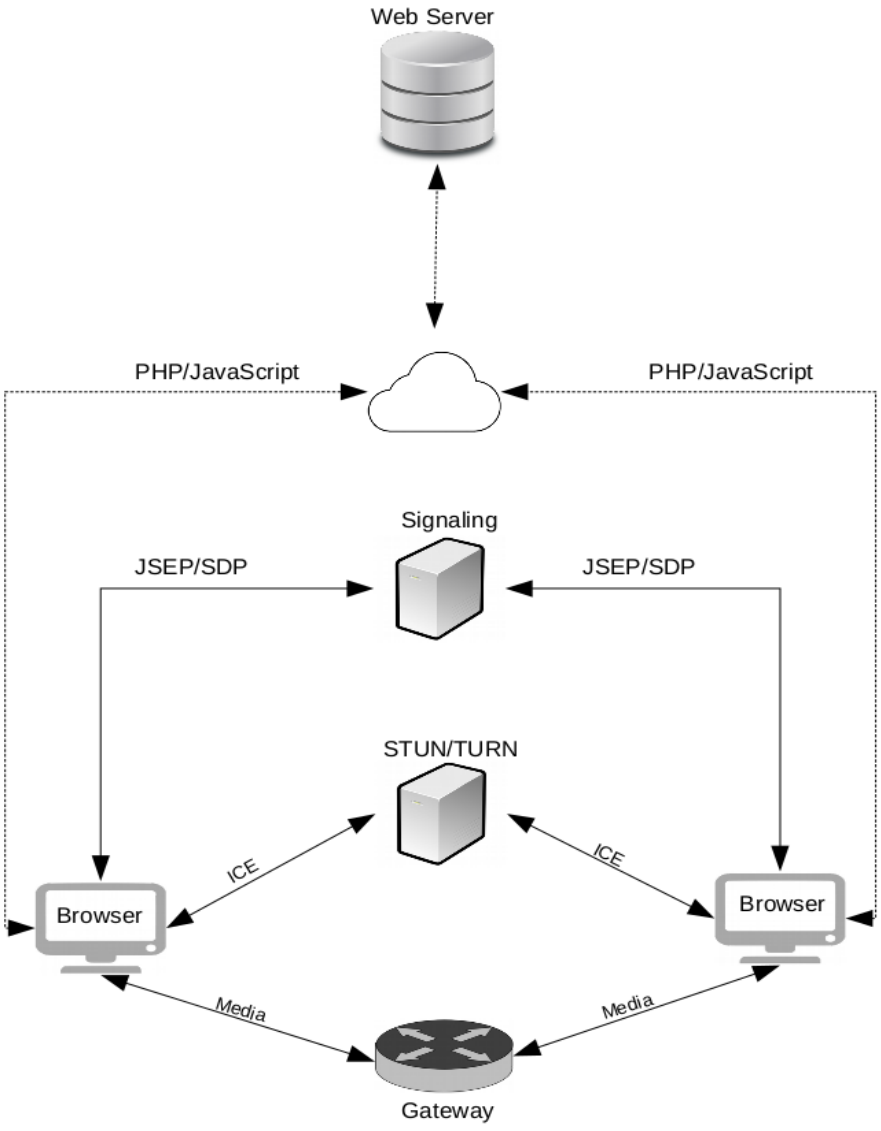

Figura 3.1: Modelo geral do funcionamento do projeto

3.2.2 Base de dados

Apresenta-se, na figura 3.2, o diagrama Entidade Relacionamento da base de dados utilizada para suportar a implementação das salas imersivas.

A base de dados representada pelo diagrama de entidades e relacionamentos possui as seguintes tabelas:

- cookies
- tokens
- feedbacks
- ratings
- logs
- friends
- users
- calls
- manages
- makes
- screens
- short_urls
- rooms

3.3 Estado inicial

Como já referido anteriormente, a plataforma de Salas Imersivas WebRTC foi promovida pela FCCN, com o seu desenvolvimento a iniciar-se no ano letivo anterior por parte de

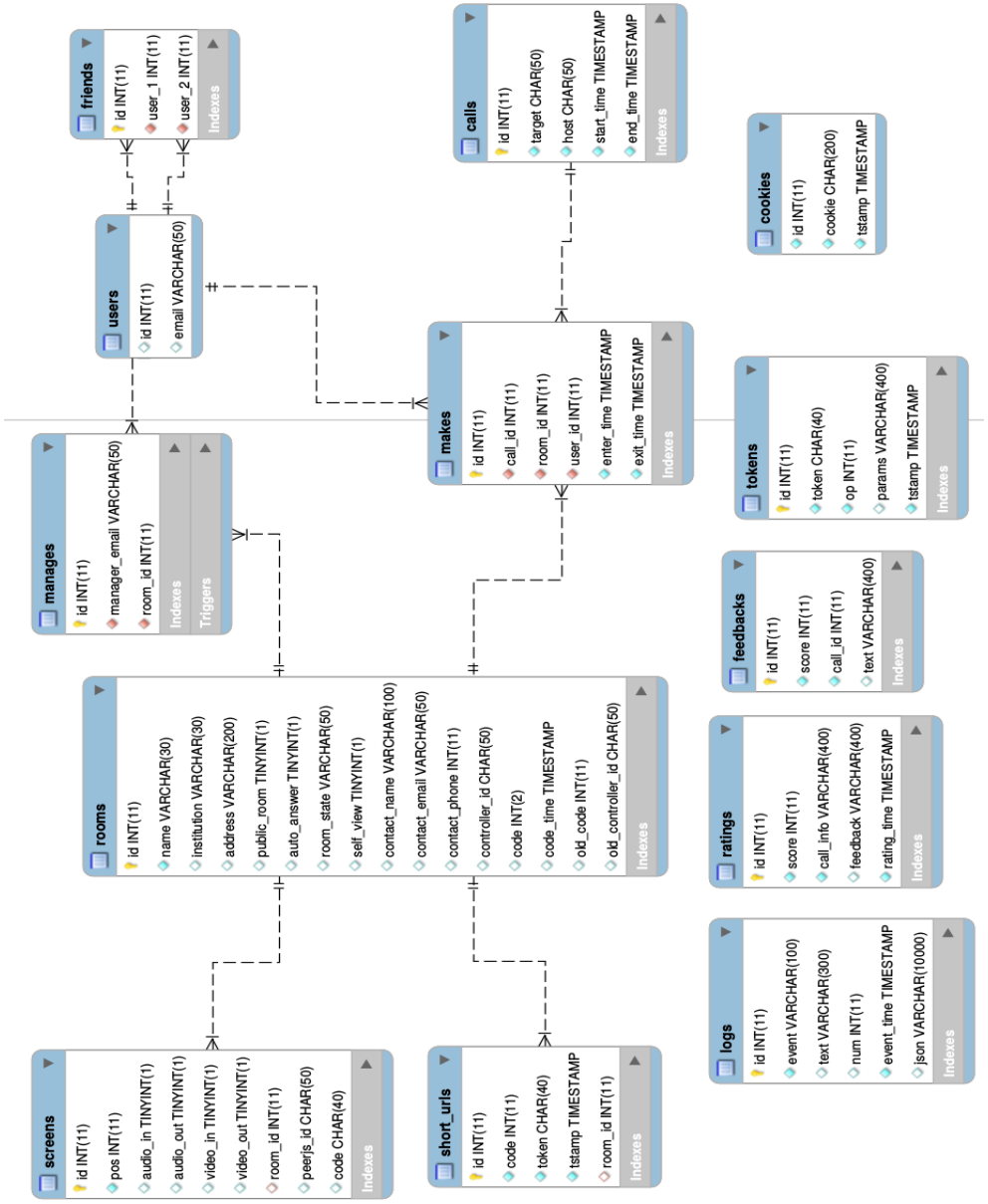

Figura 3.2: Diagrama ER

dois estudantes de uma instituição universitária de Lisboa, que foram os responsáveis pela primeira fase de desenvolvimento desta solução, implementando as características básicas para tornar a plataforma funcional, nomeadamente a realização de uma chamada entre utilizadores registados nesta mesma plataforma.

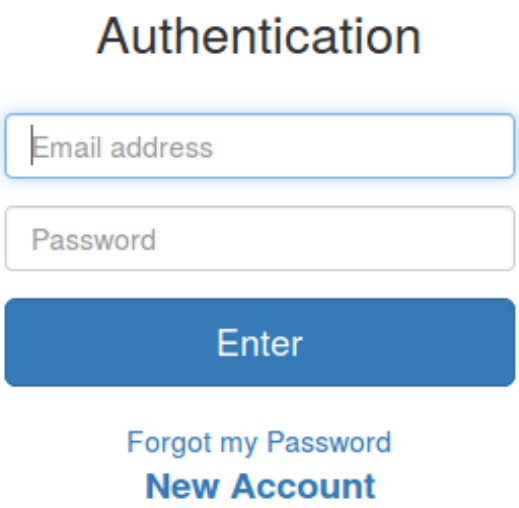

Figura 3.3: Login na plataforma

3.3.1 Autenticação

A plataforma conta já com um processo de registo e autenticação de utilizadores onde, através da criação de um utilizador, é possível definir informação relativa ao mesmo como o nome, a organização à qual pertence, o email que queira utilizar para autenticação e a correspondente palavrapasse (figura 3.3). Essa informação é armazenada numa base de dados MySQL para que futuramente possam ser executadas *queries* para obter informação para *login* ou outros processos que possam envolver tais informações.

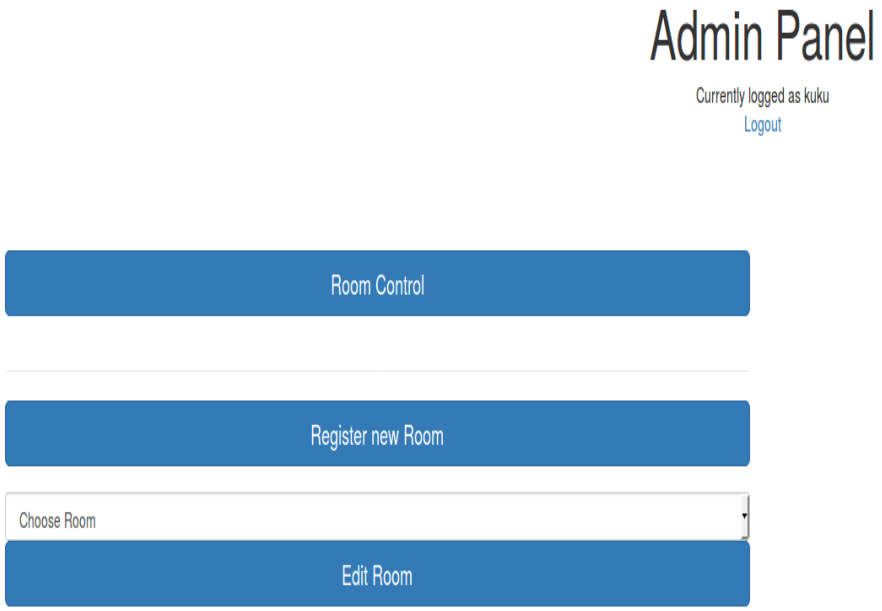

Figura 3.4: Painel de gestão

3.3.2 Gestão de salas

Após o registo e autenticação bem sucedidos, o utilizador é redirecionado para um *dashboard* onde conta com algumas opções relativas à gestão das salas. Estão disponíveis opções como: controlar uma sala, criar uma nova sala ou editar uma sala existente (figura 3.4).

Controlar Sala

Nesta secção é possível controlar uma sala existente através de um código que pertence à correspondente sala, código que vem num formato numérico, gerado automaticamente a quando da criação da mesma(figura 3.5).

Registar Sala

Para o registo de uma nova sala é necessário o preenchimento de um formulário fornecendo informação relativa ao nome, endereço, instituição, foto da sala, nome, email e telefone

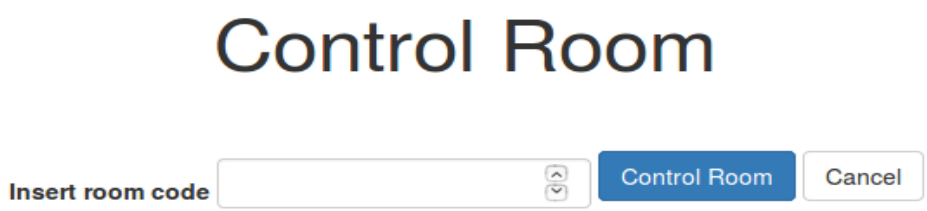

Figura 3.5: Acesso ao controle de uma sala

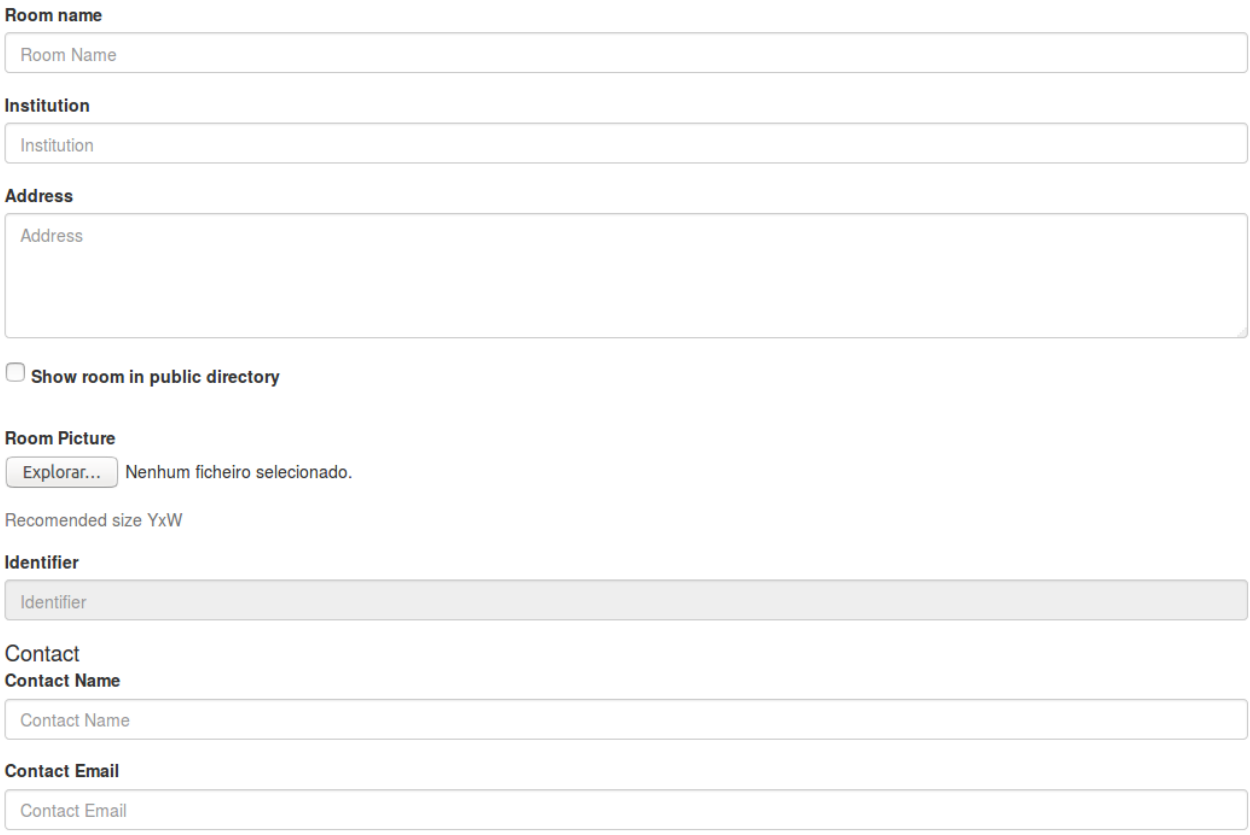

Figura 3.6: Formulário de informação da nova sala

de contacto e uma opção para deixar a sala visível publicamente ou não (figura 3.6).

Após preencher o formulário, é reencaminhado para a sala onde é possível editar e adicionar ecrãs que por ventura estarão disponíveis para acesso durante a chamada (figura 3.7).

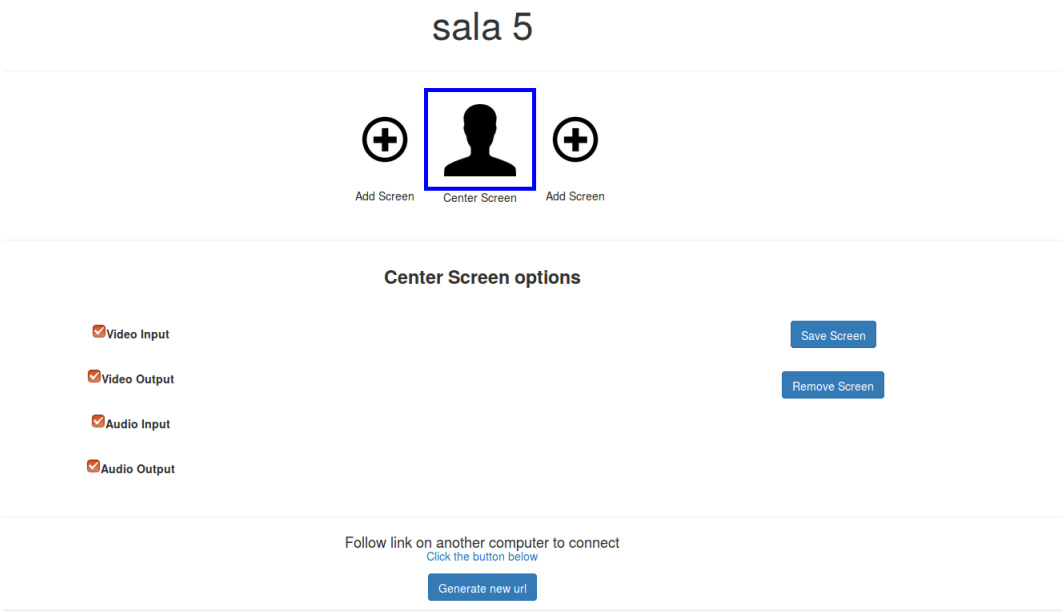

Figura 3.7: Criação/edição do ecrã para a chamada

Após confirmada a criação do ecrã, é possível por fim iniciar a sala e ficar à espera de uma chamada ou chamar outra sala. Também é disponibilizado um URL curto para facilitar este processo com as mesmas definições na próxima vez que quiser iniciar esta sala (figura 3.8).

Por fim, temos o diretório de salas e a informação relativa às mesmas (figura 3.9).

Para concluir o processo, resta apenas escolher uma sala que esteja disponível (online) e clicar no botão para fazer a chamada figuras (3.10) e (3.11).

Editar sala

Na edição de uma sala, a única diferença em relação ao formulário de criação de uma nova é a adição de uma opção para atendimento automático de chamadas na mesma (figura 3.12).

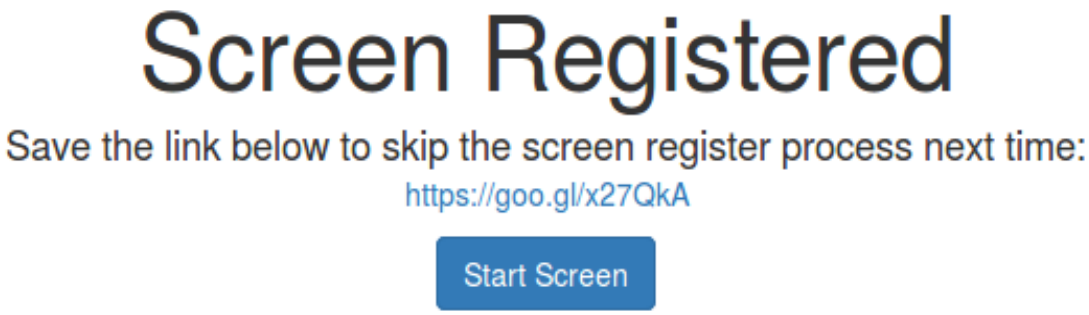

Figura 3.8: Sala pronta a ser iniciada

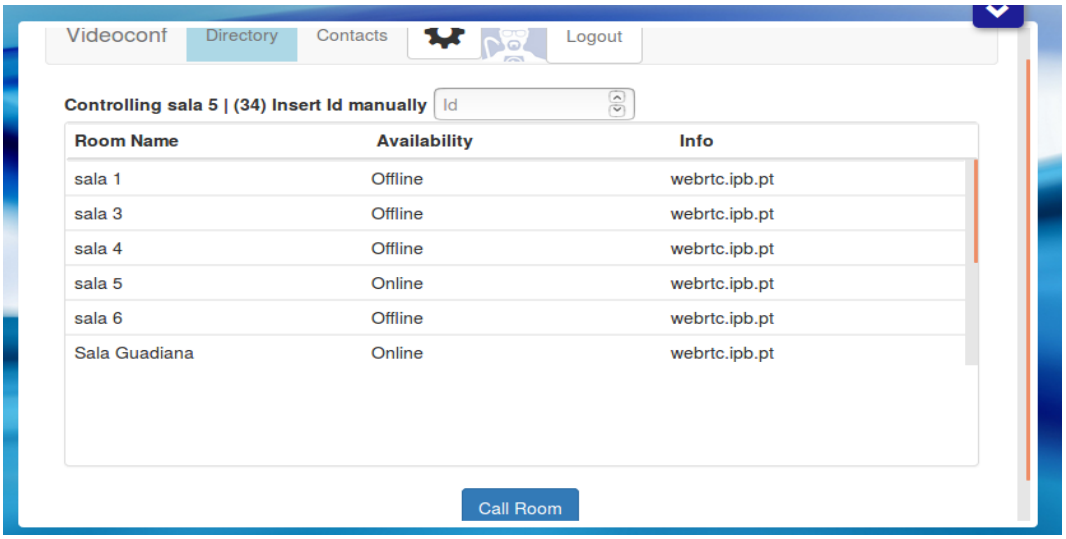

Figura 3.9: Diretório de salas existentes no domínio

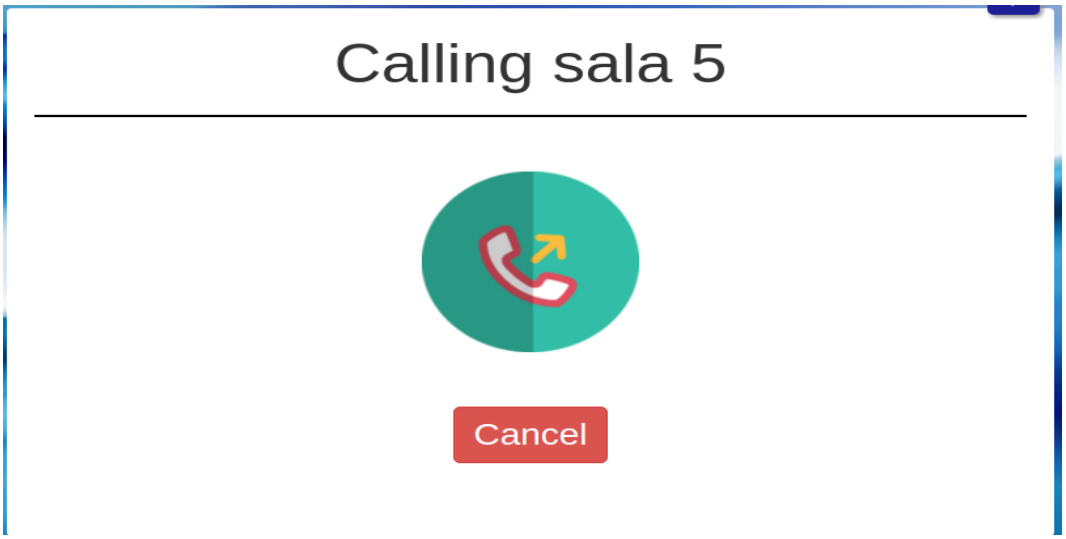

Figura 3.10: Pedido de chamada

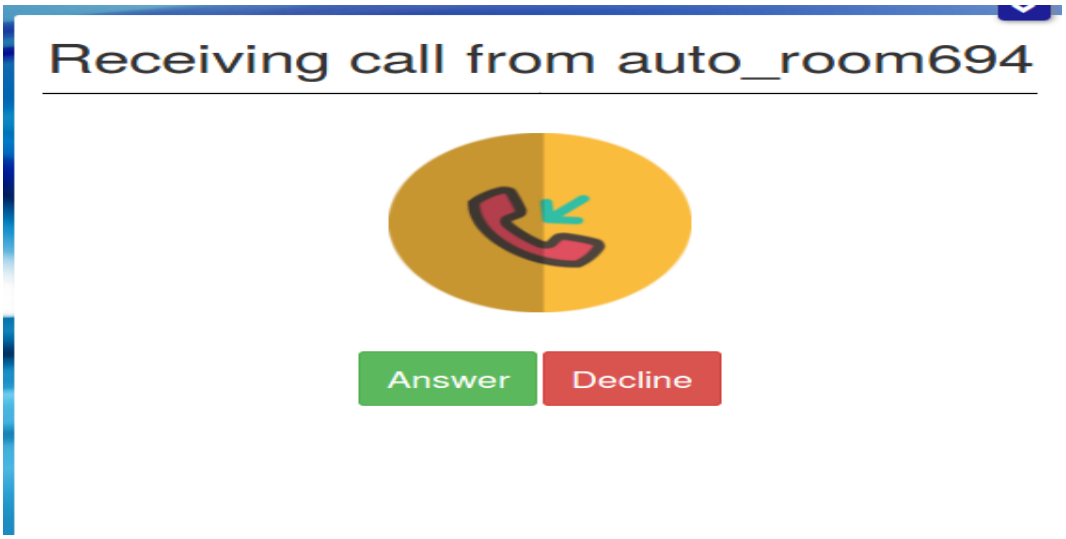

Figura 3.11: Indicação de receção do pedido de chamada por parte do alvo

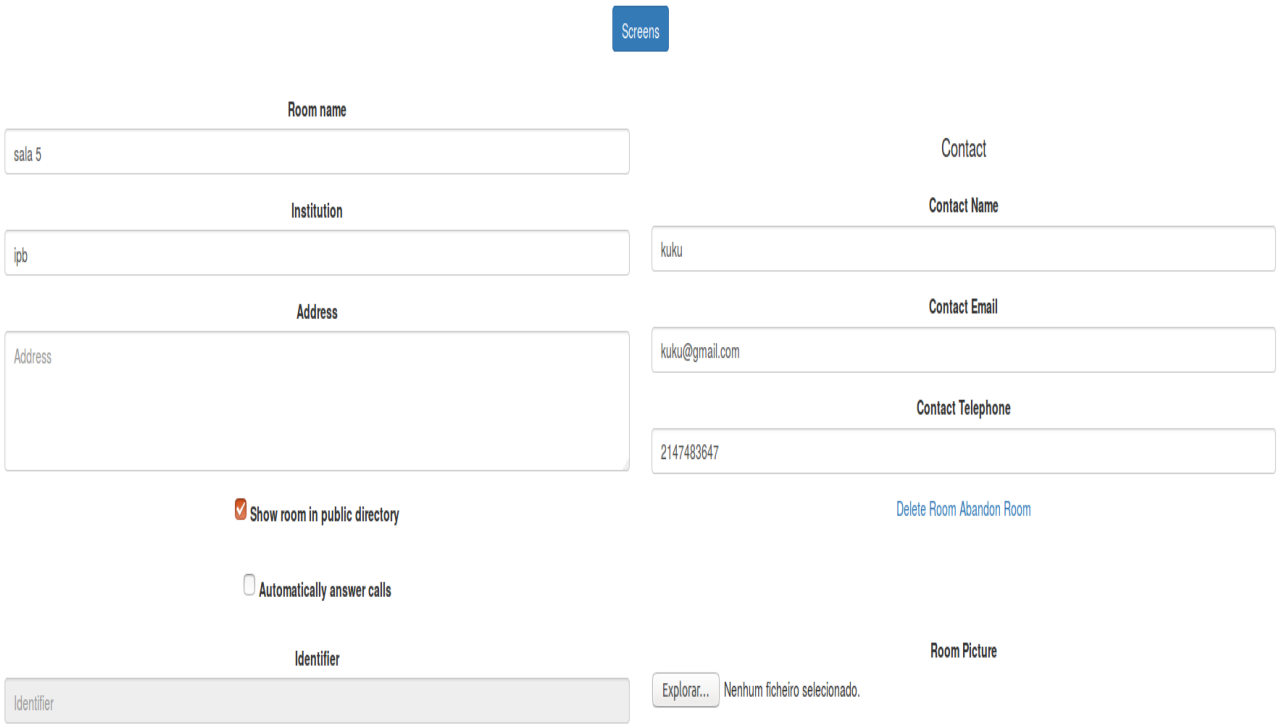

Figura 3.12: Edição de uma sala

3.3.3 Exemplo de operação

Com base na descrição da plataforma feita anteriormente, assim temos a possibilidade de realizar uma vídeo-chamada por duas formas diferentes, partindo do pressuposto que o alvo da chamada se encontra pronto para receber a mesma:

Utilizador Ad hoc

Para estabelecer uma chamada utilizando este método, basta ir na página inicial da plataforma e seguir para "instant room", onde serão criadas automaticamente as condições para ter uma sala e um ecrã válidos, para que este possa ir diretamente ao diretório de salas e ligar ao outro participante da chamada.

Utilizador registado

A outra opção para realizar uma chamada é através da criação de um utilizador permanente, onde é necessário fazer o registo do mesmo na página inicial.

Após estar registado este tem que criar uma sala, fornecendo informação acerca da mesma.

Depois de criado, o utilizador tem que iniciar a sala para ter acesso ao diretório de salas existentes no domínio e proceder à proposta de chamada para o destino da mesma.

Capítulo 4

Trabalho desenvolvido

Após alguns meses de estudo e teste da plataforma para tentar entender as bases de funcionamento da mesma e as ferramentas e mecanismos que a compõe e que tornam possível a implementação da tecnologia WebRTC, tornou-se possível passar à fase de implementação das alterações definidas, para corrigir falhas, melhorar processos e adicionar as funcionalidades que foram propostas.

Até ao momento, o foco do trabalho tem sido a implementação das novas funcionalidades e melhorias na disponibilidade, mais concretamente a capacitação das salas imersivas como uma plataforma multidomínio e melhoria na fiabilidade da plataforma e seus componentes, bem como a adição da capacidade de recuperação da mesma em caso de falhas.

Para além das melhorias mencionadas acima, foram necessárias outras alterações tanto no código como em configurações para tornar possível a implementação das mesmas. Logo à partida foi percetível uma fraca resolução de vídeo nas chamadas o que por vezes pode tornar-se um incómodo visual com ecrãs de maior tamanho. Foi possível corrigir isso, através do aumento do bit-rate nas configurações do Janus passando do valor inicial de 128000 para 1024000, o que garante uma grande melhoria na qualidade de vídeo, embora essa melhoria apenas será notada caso os participantes da chamada possuam dispositivos equipados para transmitir vídeos de alta resolução e ligação Internet com largura de banda suficiente para tal.

No ficheiro que contém o *script* SQL que serve para gerar a base de dados, chamado

"make_database.sql", é necessário adicionar uma linha logo após a entrada "CREATE" para criar a base de dados. É necessário indicar que a base de dados a utilizar para aplicar as *queries* que vêm a seguir é a criada logo antes, caso contrário existe possibilidade de comportamentos diferentes, consoante o motor de base de dados. No "mysql client", a ausência desta linha resulta na interrupção da execução do script com uma mensagem de erro com a linha ao qual se refere. Já no "mysql workbench", caso estejam presentes mais que um "schema", quando executado este *script* sql, a base de dados é criada, mas as tabelas que deveriam pertencer a esta, são criadas na base de dados que se encontra definida como base de dados padrão.

Foram identificados alguns erros no HTML, que embora não fossem nucleares ao funcionamento da plataforma, ainda assim foi necessário corrigi-los. De notar que algumas secções de código que se encontravam comentados por ter sido encontrado outra solução ou terem sido melhorados, foram apagadas.

4.1 Multidomínio

4.1.1 Lógica de funcionamento

Tornar a plataforma multidomínio foi o primeiro e mais complexo desafio no desenvolvimento da plataforma a partir do estado em que nos foi entregue. Isto porque esta evolução pressupõe o estudo e utilização de outras tecnologias que acabariam por entrar em ação neste escopo de salas imersivas multidomínio. Nomeadamente o DNS_SRV foi o agente catalisador para que pudesse ser possível estabelecer a ligação entre salas em máquinas diferentes e não relacionadas. Este processo será descrito mais à frente.

Para poder entender o que se pretende dizer com a palavra multidomínio é necessário contextualizar o projeto e os seus objetivos. Imagine-se a existência de uma sala "A" criada na instância das salas imersivas que responde pelo endereço "www.instancia-a.com", e uma sala "B" que foi criada numa outra instância da plataforma de salas imersivas respondendo pelo endereço "www.instancia-b.gov". O objetivo é fazer com que a sala A

consiga estabelecer uma chamada com a sala B.

A verdade é que esse pressuposto desta forma parece simples, porém levanta inúmeras questões antes mesmo de pensar em sua implementação. Dividindo o problema por perguntas, a primeira seria: como é que a ligação seria feita? Da forma como se encontra implementada, existe uma comunicação utilizando peers para negociação dos termos da chamada. Como esta negociação iria decorrer se os peers são criados automaticamente em cada instância quando se acede ao diretório de salas existentes? Ao criar um peer, é-lhe atribuído um "id" que serve de identificador único. A utilização de um único peer-server surge logo à partida como solução. Assim estes peers seriam criados em mesmo lugar, logo teriam conhecimento de todos os criados no mesmo peer-server. Esta solução acaba por ser invalidada por alguns motivos.

A primeira razão seria a escalabilidade limitada, pois tornando o serviço de peers um serviço centralizado, implicaria garantir que todas as instâncias instaladas em qualquer domínio tivessem garantia de acesso para criar os peers que representariam as suas salas e os seus ecrãs. Se, pela simplicidade do exemplo, assumirmos que uma chamada envolvendo 3 instâncias com 3 ecrãs cada, tal representaria um total de 12 peers necessários para uma simples chamada. Não é difícil de identificar que esses números, num ambiente real onde estariam inúmeras instâncias conectadas ao mesmo tempo com inúmeras salas e ecrãs, exigiria um poder de processamento enorme por parte do peer-server, o que podia traduzir-se em custos de infraestrutura maiores para não correr o risco de ter um problema com o tempo de resposta da plataforma. Sem mencionar que, em caso de problemas, tendo um único ponto de falha é muito arriscado, o que podia comprometer o sistema todo.

Então a solução passaria por deixar cada instância lidar com o seu peer-server, mas voltariamos ao problema inicial. Como os peers saberiam da existência dos peers criados em outros peer-servers? Esta abordagem acaba por tornar-se um beco sem saída pois simplesmente peers criados em peer-servers diferentes não conseguem comunicar-se entre si, então a solução passa por criar os peers no mesmo peer-server, com isso parece que entramos num ciclo infinito entre estas soluções erradas.

A solução utilizada para desfazer esse impasse foi criar os peers respetivos à chamada

para fora do domínio, no peer-server do destino da chamada. Como a criação dos peers ocorre no processo antecedente à chamada, é necessário fazer uma transição de modo a preparar a chamada com as configurações para ser uma chamada para um contacto que esteja em outro domínio.

Para transferir a chamada é necessário ter informação relativa ao peer-server que se quer utilizar e que passa a ser o alvo da chamada. Essa informação tem que ser fornecida pelo alvo que, assumindo que existe, este será inquirido acerca do seu peer-server e irá responder com o endereço, porto e caminho para ter acesso ao mesmo. Assim quando obtiver essa informação, transfere a sua sessão para a instância remota das salas imersivas podendo assim começar a negociar os outros termos necessários para o estabelecimento da chamada.

O mesmo problema se apresenta após a fase de negociação, onde já está tudo a postos para iniciar a troca de media e é necessário conectar-se ao *gateway* WebRTC (Janus). Mas a mesma solução aplicada no problema dos peers, serve como solução para este caso. Assim, da mesma forma, o alvo será inquirido acerca da instância de Janus que pretende utilizar para estabelecer a chamada e a resposta é entregue da mesma forma, podendo assim finalmente estabelecer a chamada e trocar media.

Esta chamada é estabelecida partindo de um pressuposto que assume o conhecimento da existência do alvo da chamada, mas é facilmente percetível que isto pode não corresponder à verdade, e há que avaliar todos os casos possíveis.

A segunda questão que se levanta é: como saber se o alvo esteja online ou este sequer exista. Dessas dúvidas surge a necessidade de um método de descoberta e pesquisa de contactos. Uma instância tem que ser capaz de inquirir outras instâncias acerca dos utilizadores registados que possui. Através desta necessidade, surgiu a ideia de utilizar registos DNS para ultrapassar os problemas de descoberta entre as instâncias, utilizando o tipo de registos SRV que são específicos para esse efeito.

Cada instalação das salas imersivas passa a pressupor pelo menos uma entrada DNS do tipo SRV que responda com o endereço de onde se encontra o serviço disponível. Na

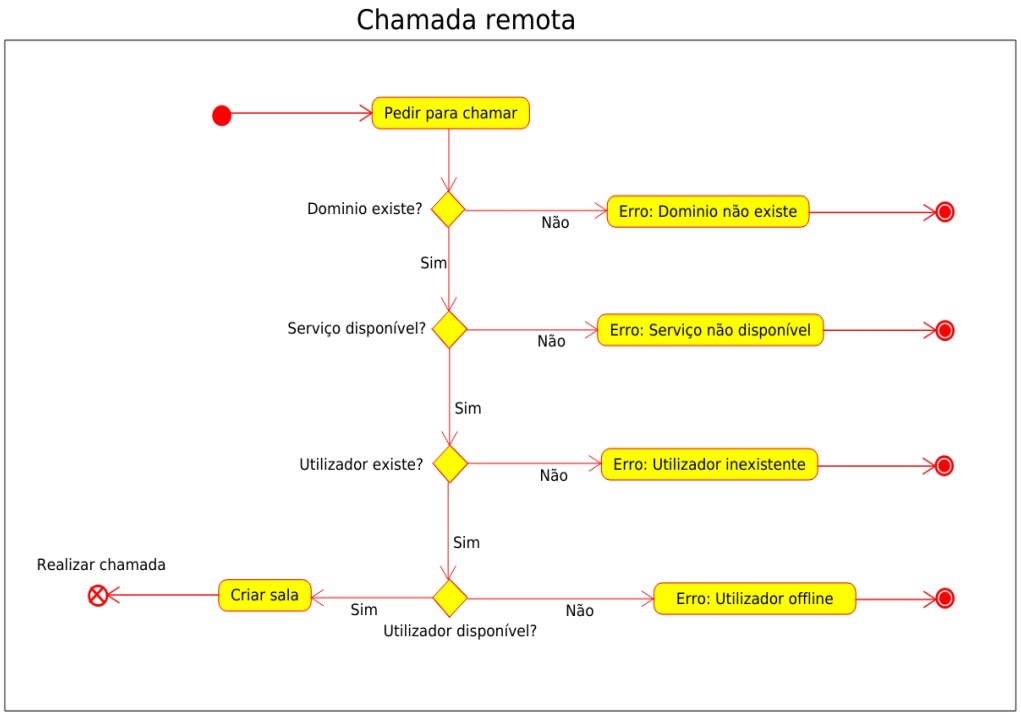

Figura 4.1: Diagrama de atividade de uma chamada remota

prática, foi implementado da forma que quando alguém queira ligar a uma sala, é necessário dar o endereço completo do alvo no formato nome do alvo, seguido de um "@" e o domínio onde este se encontra. Por exemplo se quiser ligar ao "JohnDoe" que se encontra alojado na instância "exemplo1.com", teria de ser fornecido no formato "johndoe@exemplo1.com". Quando clicar no botão para ligar, seria feita a *query* DNS para o endereço "exemplo1.com" a perguntar se o domínio existe. Caso exista, se possui o serviço WebRTC disponível, se possuir então responde com o endereço e porto de onde este serviço está disponível. Este processo assemelha-se muito aos registos MX utilizados nos serviços de correio eletrónico.

Quando tiver informação acerca da disponibilidade do WebRTC, são feitas mais verificações, relativas a dados que se encontram na base de dados, ou seja, serão feitas *queries* ao MySQL que se encontra associado à instância das salas imersivas onde o alvo se encontra instalado. Primeiro, para saber se o utilizador existe. Caso exista, também é verificado se

este se encontra online ou não. Caso esteja online, por fim, é criada uma sala na instância remota das salas imersivas e esta salas recebe os seus ID's dos peers relativos à sua sala e os seus ecrãs, para de seguida enviar a proposta de chamada para o alvo. O diagrama de atividades na figura 4.1 representa este processo de estabelecimento de uma chamada para um utilizador que esteja num domínio diferente.

4.1.2 Implementação prática

Tendo em consideração a descrição feita anteriormente no ponto de vista teórico, a implementação das alterações necessárias para possibilitar as chamadas multidomínio envolvem a adição e edição de muitas linhas de código, sobretudo PHP e JavaScript, pois foram necessárias inúmeras adaptações para que tanto a descoberta do alvo da chamada como o estabelecimento da ligação ao mesmo fosse possível. Comecemos por descrever o registo DNS SRV que foi utilizado para descobrir o alvo da chamada na Internet.

DNS SRV Resource Record

A este projeto, interessa o resource record DNS "SRV" que é utilizado para especificar a localização de serviços. É fácil de perceber onde este tipo de RR se encaixa no escopo do projeto, tendo sido proposta a sua utilização para a descoberta do serviço webrtc utilizando um *full domain name* [28].

Para tirar partido da funcionalidade dada pelo DNS SRV, todos os *hosts* que possuam instalado uma instância das salas imersivas são obrigados a adicionar uma entrada deste tipo no ficheiro da zona de seu respetivo DNS *server*, para que seja possível identificar a disponibilidade do serviço WebRTC. Primeiro é preciso entender o formato de uma entrada SRV no DNS e como esta entrada será útil para a descoberta do serviço [29].

Segundo o RFC 2782 [28], o formato definido para as entradas SRV segue o seguinte padrão:

- **Service** – Começa sempre por um "_" (underscore) para evitar colisões com outras designações DNS; segue-se o nome do serviço em questão e é terminado por um

"."(ponto) para definir o fim da designação do serviço.

- **Protocol** – Segue-se a designação do protocolo que também começa por um "_", seguido do nome do protocolo, que por sua vez, à semelhança do serviço, é terminado por um ".".
- **Name** – Segue-se o nome do domínio ou zona ao qual este RR se refere, e também é terminado por um ".".
- **TTL** – O ttl segue a mesma definição standard aplicado aos RR.
- **Class** – Também segue o standard RR.
- **Priority** – Campo numérico inteiro que define a prioridade de um host, pela qual os clientes podem tentar contactar. O cliente deve escolher sempre o host com o valor mais baixo neste campo. Caso estes encontrem *hosts* com o mesmo valor neste campo, então passam a decisão para o próximo campo, que determina o peso.
- **Weight** – Mecanismo de escolha caso existam vários *hosts* com o mesmo valor no campo da prioridade. Nestes casos, deve ser escolhido o host com maior valor no peso.
- **Port** – O porto onde se encontra disponível o serviço no host que o fornece.
- **Target** – O nome do domínio onde o serviço pretendido está disponível. Caso o valor neste campo seja um "."(ponto), significa que o serviço não está disponível neste domínio.

A entrada que foi concebida para utilizar este tipo de registo DNS e que foi implementada no ambiente de teste e consequentemente testada com sucesso é da relativa fácil compreensão após a perceção do formato SRV e de como funciona:

```
_multiconf._webrtc.webrtc.ipb.pt.  86400 IN SRV 10 1 433 webrtc.ipb.pt.
_multiconf._webrtc.flintheart.estig.ipb.pt.  86400 IN SRV 10 1 433 flintheart.estig.i
```

Utilizando a ferramenta linux "nslookup" no terminal e especificando o tipo de entrada que

procuramos no DNS com a opção -q", dá para confirmar que os registos foram adicionados ao ficheiro relativo a esta zona "ipb.pt".

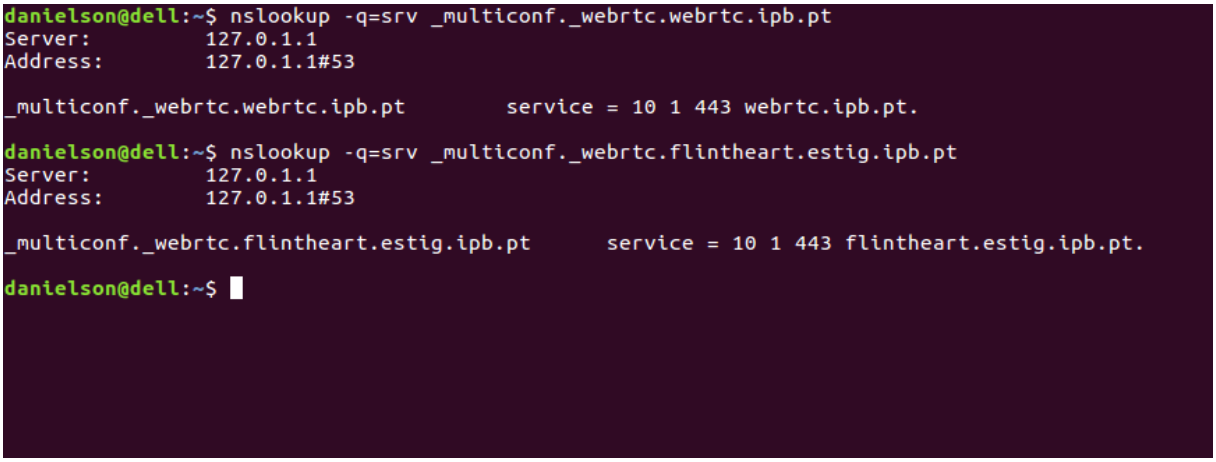

Figura 4.2: Verificando a existência do registo SRV com nslookup

Com os registos SRV adicionados e operacionais, fica assim possível criar código PHP funcional para descobrir se determinado host existe e fornece o serviço WebRTC. Para tentar estabelecer uma chamada para uma sala em outro domínio, primeiro é preciso saber da existência do mesmo. Com isto em mente foi adicionada a funcionalidade de chamada para uma sala em outro domínio através do RR do alvo da chamada. É possível utilizar esta nova funcionalidade tanto a partir do *dashboard*, como também estando no diretório das salas disponíveis (figura 4.3 e figura 4.4).

À semelhança dos registos MX do serviço de correio eletrónico, esta funcionalidade foi implementada visando interpretar o nome do contacto na sua totalidade, particionando o mesmo em secções para poder aplicar as verificações necessárias antes de tentar estabelecer conectividade entre os dois pontos da chamada. Quando é introduzido um contacto e consequente clique no botão para realizar a chamada, o conteúdo é dividido em duas partes utilizando os métodos JavaScript "substring" e "lastIndexOf".

Substring é um método que permite extrair carateres de uma "string" utilizando como argumentos um "start" e um "end" para definir o início e fim da string que se pretende obter a partir da original.

LastIndexOf por sua vez permite retornar a posição da ultima ocorrência de um

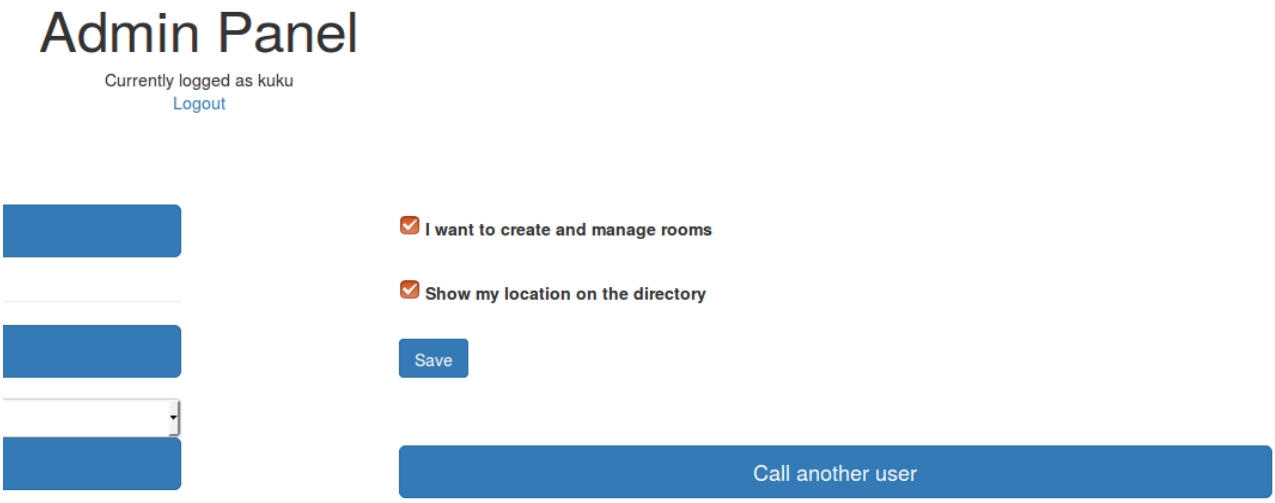

Figura 4.3: Botão para realizar conectar-se a utilizador remoto no dashboard

caractere ou string em uma string específica.

Foi utilizado o "lastIndexOf" para procurar o "@", que serve como ponto de divisão entre um nome e um domínio quando se trata do "full name" que é introduzido, sendo de seguida guardado o nome do utilizador numa variável e o domínio noutra variável.

Utilizando o **AJAX**, é possível enviar o conteúdo da variável "domínio" para "controller/dnsquery.php" onde serão feitos os testes de verificação relativos ao domínio.

Em primeiro lugar, é verificado o conteúdo da variável enviada. Caso a mesma esteja vazia, as verificações são interrompidas e é retornada uma mensagem de erro.

Caso passe a verificação anterior, é testada a existência do domínio. Esta verificação é feita utilizando o método PHP **checkdnsrr**. Este método recebe dois argumentos, onde o primeiro é uma string com o domínio a verificar, o segundo é o tipo de **RR** a que este domínio se encontra associado. Se o valor retornado por este método for falso, a execução é interrompida com uma mensagem de erro. Caso seja diferente, avança-se para o próximo

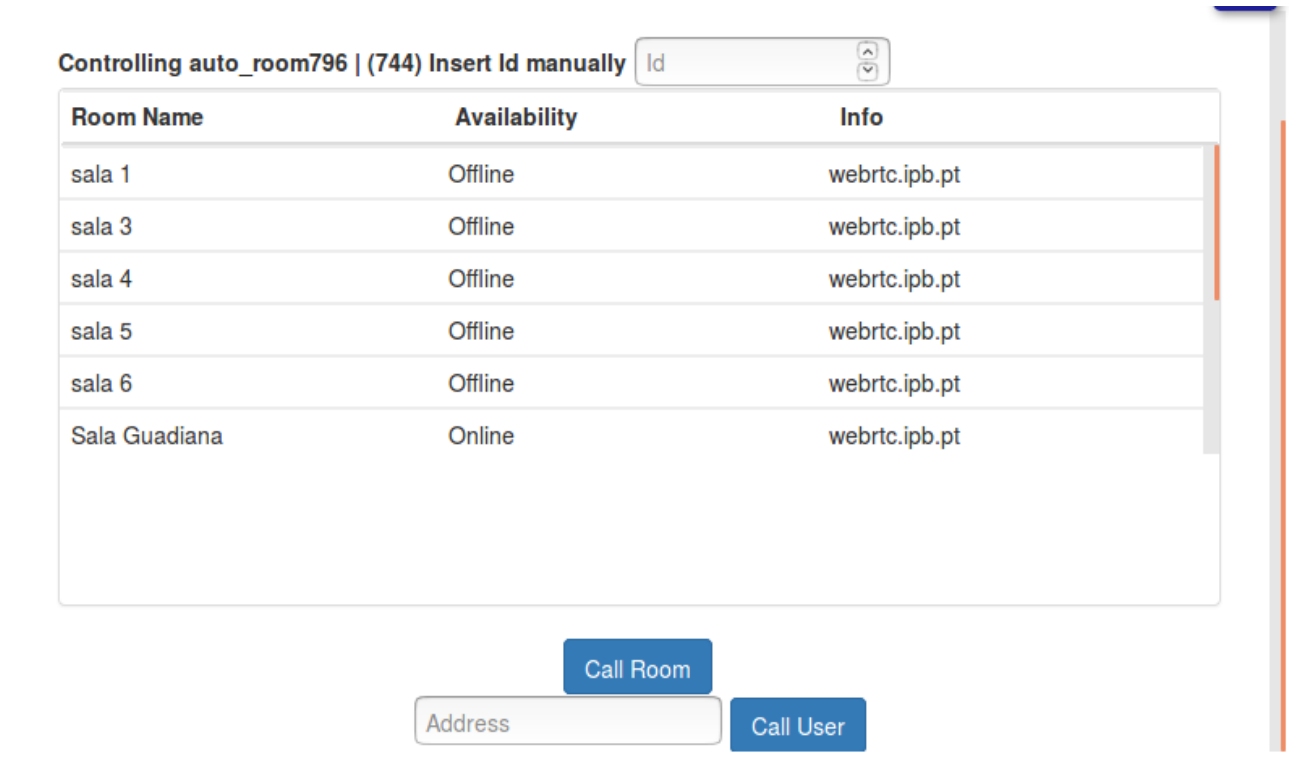

Figura 4.4: Botão para realizar conectar-se a utilizador remoto no diretório

passo.

De seguida, o ao serviço WebRTC faz uma verificação utilizando o método PHP **dns_get_record**, que é um método que através de um dado *hostname* e um tipo de RR, retorna os dados do RR associado a este *hostname*. Se o valor retornado por este método for vazio, a execução é interrompida, caso contrário é retornada uma matriz com os dados completos do RR, que são armazenados numa variável.

Por fim é retornado o valor final pretendido através do método **json_encode**, que, se todas as verificações forem bem sucedidas, consiste numa string contendo o endereço de onde encontrar o serviço pretendido, que no caso é o WebRTC.

Quando é obtido o endereço do host onde se encontra o WebRTC disponível, são então transferidos os dados para realizar a chamada para a sala remota. Esta "transferência" consiste em redirecionar a página atual para o endereço remoto onde se encontra alojada a sala alvo da chamada (este processo será descrito mais à frente) com dados como o domínio

da instância das salas imersivas onde a sala é originária, o nome do utilizador que controla esta mesma sala, e o nome do utilizador que controla a sala alvo da chamada. Este redirecionamento é feito utilizando o método **top.location.href**. Tal acontece porque são utilizados iframes na incorporação do diretório das salas na página, ou dito de uma outra forma, o *controller* consiste em um iframe que se encontra dentro do *slave*, dai a necessidade do "top" antes do "location.href".

Antes de chegar ao ponto onde estão as duas salas no mesmo diretório e prontas para a chamada, são feitos alguns saltos entre páginas para preparar as condições para tal, utilizando os dados que foram enviados no URL. O primeiro salto é do diretório anterior ou página "chamar pessoa" para o "auto_room_rem.php", que consiste numa adaptação do "auto_room.php" de forma a que seja possível criar uma sala num servidor remoto com dados válidos. Criar a sala localmente torna-se simples já que basta utilizar-mos os valores guardados na sessão do browser. Como não é possível passar a sessão de um domínio para outro, os dados tem que ser transferidos e interpretados de uma outra forma. Essa passagem de dados é feita utilizando as variáveis globais que o PHP dispõe, nomeadamente com o "$_GET" ou "$_POST".

Neste ponto, são criadas a nova sala e o ecrã que pertence a essa sala na base de dados remota, onde a sala alvo da chamada também se encontra registada, passando por algumas verificações necessárias. Quando prontos, é feito outro salto com os dados resultantes do processo anterior. Este salto é feito para o "html/register_screen_rem.php" que também foi criado com o intuito de contornar o problema com as sessões em servidores diferentes. Outra vez utilizando as variáveis globais do PHP, são passados valores de variáveis para esta página para confirmar que a sala e o ecrã foram criados de forma correta. Sendo verdade, dá-se o último salto, através de um redirecionamento para o "controller/slave_rem.php" com algumas variáveis necessárias.

Tendo criado a sala e seu ecrã e guardando estes na base de dados, resta apenas preencher a informação em falta relativa à sinalização através de peers. Assim que a página do *slave* é carregada, também é carregado o *controller*, que é incorporado como um "iframe" do *slave* e que contém o diretório das salas. Automaticamente são executadas

funções javascript que provém do "controller/resources/javascrpit/slave_rem.js.php", que criam os peers necessários e introduzem os peer id's para a base de dados, através de funções **AJAX**, onde as salas e ecrãs estão registados para utilizar os peer id's como identificadores destes.

Feito isto, e estando na mesma plataforma que a sala alvo, podemos considerar que a chamada está a apenas um clique de distância para ser estabelecida.

A plataforma foi alterada com o objetivo de permitir chamadas multidomínio e para que isso fosse possível, foi necessário fazer com que as partes relativas a esta funcionalidade fossem criadas com características genéricas. Genéricas do ponto de vista em que quando uma pessoa(sala) tenta contactar uma outra que se encontra noutro servidor, a única informação que possui do alvo da chamada é o seu *full name*. O resto da informação relativa à localização dos serviços necessários que servem de suporte à chamada são incógnitas num primeiro instante, sendo descobertas posteriormente após verificações. Da forma como a plataforma funcionava anteriormente, uma sala só conseguia ligar-se a uma outra se estivessem ambas na mesma instância das salas imersivas, pois a localização dos serviços encontram-se *hard coded*. Assim são apenas utilizados os serviços que se encontram nestes endereços, impossibilitando a comunicação entre peers, conexão Janus ou descoberta de candidatos ICE pelo Turnserver. Tal criou a necessidade de tornar a funcionalidade genérica, para qualquer que seja a localização dos serviços necessários, tornando o processo de estabelecer chamada num processo transparente e automatizado.

4.2 Fiabilidade

Inúmeras foram as vezes que a meio de uma chamada acontecia uma perda de conexão, onde a chamada bloqueava e já não era possível comunicar-se com os outros participantes. Este problema deve-se à fraca fiabilidade dos serviços que dão suporte às salas imersivas, com especial ênfase no Janus Gateway. Quando tais paragens acontecem e utilizando a ajuda dos ficheiros de *log* destes serviços e também o depurador do browser, é possível perceber qual dos serviços causou este *crash*.

Porém, a ocorrência destes *crash's* não estão limitados apenas ao período ativo de uma chamada. Foi notado também com o sistema fora do contexto da chamada, podendo simplesmente parar um ou todos os serviços adjacentes às salas imersivas, o que dificulta o processo de descoberta da causa do *crash* ou que condições causam o mesmo.

Como foi referido anteriormente, o foco está virado no Janus pois, sempre que acontece um *crash*, 90% das vezes o serviço afetado é o Janus. Foram feitos esforços com o objetivo de obter mais informação acerca da causa deste erro, efetuando numa primeira fase uma análise ao ficheiro que contém os *logs* da aplicação. É possível identificar o momento em que o erro ocorre, mas infelizmente a causa do mesmo não é apresentada. Para tentar ir mais além na procura da raiz da causa do *crash*, foi instalado o depurador GDB com o intuito de compilar o Janus num ambiente controlado para tentar perceber melhor as causas. Após repetir a realização de algumas chamadas para tentar replicar o erro, chegou-se à conclusão que, segundo os *logs* do GDB, o problema com o Janus é um problema relativo a gestão de memória feito pelo mesmo.

Mais precisamente, o GDB mostra que, no momento do *crash*, acontece um *Segmentation fault* ou falha de segmentação(figura 4.5, figura 4.6 e 4.7), que é um erro que ocorre quando um programa tenta alocar um espaço de memória que já se encontra ocupado por outro programa.

```
[Thread 0x7f67e1ffb700 (LWP 22825) exited]
[New Thread 0x7f67e1ffb700 (LWP 22826)]
[Thread 0x7f67e1ffb700 (LWP 22826) exited]
[New Thread 0x7f67e1ffb700 (LWP 22827)]
[Thread 0x7f67e1ffb700 (LWP 22827) exited]
[New Thread 0x7f67e1ffb700 (LWP 22828)]

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7f67dcff1700 (LWP 22768)]
0x00007f683ff86d48 in g_type_check_value () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
```

Figura 4.5: Output GDB1

Após entrar em contacto com o criador do Janus, Lorenzo Mineiro, e alguma troca de informação sobre este caso em particular, o mesmo sugeriu que experimentassemos outras versões do mesmo, que estariam em outras "Pull Requests"(PR). Foi seguido o conselho do Lorenzo mas com resultados pouco diferentes dos que haviam sido conseguidos anteriormente.

```
(gdb) backtrace
#0  0x00007f683ff86d48 in g_type_check_value () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
#1  0x00007f683ff89229 in g_value_peek_pointer () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
#2  0x00007f683ff7c541 in g_signal_emitv () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
#3  0x00007f684072deb7 in ?? () from /usr/lib/x86_64-linux-gnu/libnice.so.10
#4  0x00007f6840735c1e in ?? () from /usr/lib/x86_64-linux-gnu/libnice.so.10
#5  0x00007f683fc8e613 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#6  0x00007f683fc8db6d in g_main_context_dispatch () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#7  0x00007f683fc8df48 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#8  0x00007f683fc8e272 in g_main_loop_run () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#9  0x000000000042c53c in janus_ice_thread (data=0x7f67e8022a00) at ice.c:2288
#10 0x00007f683fcb4845 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#11 0x00007f683e57e064 in start_thread (arg=0x7f67dcff1700) at pthread_create.c:309
#12 0x00007f683e2b362d in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:111
```

Figura 4.6: Output GDB2

A partir deste ponto é percetível que tínhamos um grande problema em mãos. Uma plataforma de vídeo-conferência não pode correr o risco de simplesmente cortar a conexão numa chamada, tornando-se inviável a sua utilização em casos que se procura uma maior robustez para esta tarefa.

Então, como forma de minimizar este problema, até que a sua origem seja descoberta e suprimida, foram pensadas soluções para reduzir o seu impacto. Com isso em mente foi proposta a implementação de uma solução que, após acontecimento do *crash*, faça com que os serviços sejam reiniciados para que a chamada seja retomada sem a necessidade de esperar que um administrador das salas imersivas vá reiniciar estes serviços, tornando-o assim num processo automático.

Esta solução consiste num "bash script" que fica responsável numa primeira fase pela monitorização dos serviços necessários ao funcionamento correto da plataforma. Caso estes, por algum motivo, deixem de funcionar, este *script* tem que ser capaz de inicia-los para dar continuidade ao funcionamento normal da plataforma.

4.2.1 Bash Script

Em tradução literal, *script* significa roteiro, que normalmente é uma palavra associada a artes como cinema ou teatro onde os atores utilizam os roteiros como guião para saberem o que dizer/fazer quando for a sua vez de atuar. Com isso em mente, podemos usar esta analogia no contexto *bash*. Uma *bash script* nada mais é que um conjunto de instruções ou

```
(gdb) where full
#0  0x00007f683ff86d48 in g_type_check_value () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
No symbol table info available.
#1  0x00007f683ff89229 in g_value_peek_pointer () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
No symbol table info available.
#2  0x00007f683ff7c541 in g_signal_emitv () from /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
No symbol table info available.
#3  0x00007f68407deb7 in ?? () from /usr/lib/x86_64-linux-gnu/libnice.so.10
No symbol table info available.
#4  0x00007f684073c1e in ?? () from /usr/lib/x86_64-linux-gnu/libnice.so.10
No symbol table info available.
#5  0x00007f683fc8e613 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
No symbol table info available.
#6  0x00007f683fc8db6d in g_main_context_dispatch () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
No symbol table info available.
#7  0x00007f683fc8df48 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
No symbol table info available.
#8  0x00007f683fc8e272 in g_main_loop_run () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
No symbol table info available.
#9  0x0000000000042c53c in janus_ice_thread (data=0x7f67e8022a00) at ice.c:2288
        handle = 0x7f67e8022a00
        __FUNCTION__ = "janus_ice_thread"
        loop = <optimized out>
#10 0x00007f683fcb4845 in ?? () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
No symbol table info available.
#11 0x00007f683e57e064 in start_thread (arg=0x7f67dcff1700) at pthread_create.c:309
        res = <optimized out>
        pd = 0x7f67dcff1700
        now = <optimized out>
        unwind_buf = {cancel_jmp_buf = {{jmp_buf = {140084066064128, 1999900672716982435, 1, 140084252049328, 18, 140084066064128, -1914416444261414749, -1913637097672292189}, mask_was_saved = 0}}, priv = {
            pad = {0x0, 0x0, 0x0, 0x0}, data = {prev = 0x0, cleanup = 0x0, canceltype = 0}}}
        not_first_call = <optimized out>
        pagesize_m1 = <optimized out>
        sp = <optimized out>
        freesize = <optimized out>
        __PRETTY_FUNCTION__ = "start_thread"
#12 0x00007f683e2b362d in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:111
No locals.
```

Figura 4.7: Output GDB3

comandos *bash* que são guardados num ficheiro de texto em vez de introduzidos no terminal ou shell. Este ficheiro é executado consoante o objetivo que se pretende e, dependendo do conteúdo do mesmo, para se obter determinado comportamento ou resultado.

4.2.2 Script desenvolvido

O *script* criado para dar resposta ao pretendido segue um algoritmo que facilita a perceção do mesmo.

- São guardados os valores da data e hora atual do sistema numa variável "**now**".
- Com a ajuda do comando *bash* **pgrep**, é possível procurar pelos valores (inteiro) que identificam os processos e que normalmente são identificados pela sigla PID, no nosso caso, interessa o valor dos PID's dos processos Janus,Turnserver e NodeJS, que são os serviços que dão suporte às salas imersivas webrtc. O valor resultante deste comando é guardado na variável "**SERVICE**".
- Em *bash scripts*, as condicionais "**if,then,else**" também são válidas e utilizando-as é possível testar o valor da variável **SERVICE** e, consoante o valor contido na variável, definir o comportamento a seguir. Segundo alguma pesquisa, os **PID**'s de numero menor que 300 são reservados ao Kernel, então faz sentido testar usando tal valor como condicional. Assim se o retorno do pgrep for maior que 300 quer dizer que o serviço se encontra ativo e nada é feito, caso seja inferior, é reiniciado o serviço em causa.

Adicionalmente foi ponderada uma outra funcionalidade a acrescentar a este *bash script* que se pode revelar importante quando se considera investigar as ocorrências dos erros, suas causas ou a frequência que estes acontecem. Com esse objetivo em mente, foram adicionados os comandos "exec" e "trap" para este efeito ao ficheiro que contém o *bash script* anterior, assim ficando apenas necessário definir o caminho e nome do ficheiro onde se pretenda guardar os *logs* do *script*.

As instruções para corretamente executar o *bash script* estão no apêndice "guia de instalação" neste documento.

4.3 Gravação e Playback

Até a data da escrita deste documento, vem sendo desenvolvida a funcionalidade de playback na plataforma, de modo a reproduzir vídeos de chamadas que o utilizador optou por gravar.

A plataforma já possui implementada a ferramenta de gravação, pelo que é necessário tratar dos ficheiros resultantes das gravações e escolher o melhor método para disponibilizar e reproduzir os mesmos.

Durante o processo de gravação de uma chamada são criados, num diretório (definido durante as configurações do Janus) dois ficheiros de extensão ".mjr", que consistem no áudio e vídeo pertencentes à gravação. Estes são gravados em ficheiros separados, pelo que é necessário levar em conta este facto no momento da definição da solução ideal.

Existem diversas abordagens possíveis para resolver esta questão de playback dos vídeos na plataforma, tendo sido consideradas duas para verificar qual seria a que melhor servia para o caso das salas imersivas:

Primeira abordagem

Na primeira abordagem pensou-se primeiramente na conversão dos dois ficheiros gerados durante a gravação, tirando proveito de um utilitário que o Janus contém chamado "janus-pp-rec" que serve para converter os ficheiros resultantes de gravações no Janus em formatos que possam ser utilizados em outras aplicações (formatos como *opus, wav, webm, h264 e srt*).

Tendo o áudio e vídeo em ficheiros separados, é necessário converter estes utilizando a ferramenta descrita acima. No caso, em ficheiros no formato *opus* para o áudio e *webm* para o vídeo.

Com os ficheiros resultantes da conversão, é necessário agora fundir estes num único

ficheiro para que este depois seja utilizado na plataforma para reprodução. Existe uma ferramenta chamada **FFMPEG** que está também disponível em Linux que serve para este efeito.

Depois de converter e fundir áudio e vídeo, resta apenas incluir estes na plataforma. Esta inclusão é feita utilizando um reprodutor de vídeos implementado em JavaScript à escolha de quem utilizar esta abordagem.

Esta abordagem embora pareça simples de perceber, pressupõe um problema incontornável. É que todo este processo é feito de modo manual, o que em ambiente real não faz sentido pois seria impossível dar resposta em tempo útil quando se fala de utilizadores em grande número.

Este problema pode ter sua resolução num *script bash* que faria todo este processo em *background* ao ritmo em que novas gravações fossem feitas.

Segunda abordagem

A segunda abordagem estudada e que se encontra em fase final de implementação consiste na não conversão dos ficheiros resultantes da gravação, optando por utilizar estes no seu formato original e reproduzindo-os com um outro *plugin* disponível no Janus, o *Record&Play*. Este *plugin* permite tanto gravar áudio e/ou vídeo, como também reproduzir estes, porém com algumas diferenças em relação ao *VideoRoom* que é utilizado para fazer chamadas. A primeira é que o *Record&Play* não permite fazer chamadas, logo não permite substituir o outro. Outra diferença em relação ao *VideoRoom* ocorre no momento de gravar, onde enquanto no *VideoRoom* são guardados dois ficheiros de extensão ".mjr" que representam o áudio e vídeo desta gravação, o mesmo acontece quando é gravado utilizando o *Record&Play*, com a diferença de, para além destes dois ficheiros, é adicionalmente criado no mesmo diretório que estes um ficheiro no formato **INI** de extensão ".nfo" que contém informações acerca do que foi gravado, nomeadamente:

- ID do handle - que serve como elemento identificador de cada par (áudio/vídeo);
- Nome do par;

- Data da gravação;
- Caminho e nome do ficheiro áudio;
- Caminho e nome do ficheiro vídeo.

Como o plugin lista os pares de ficheiros utilizando informação que vai buscar dos seus ficheiros de informação correspondentes (.nfo), logo, este não lista os criados no plugin *VideoRoom* e de momento este é foco de trabalho: capacitar o plugin *VideoRoom* de modo a que este crie, para cada gravação, o seu correspondente ficheiro "*.nfo" para poder estar disponível no *plugin* de reprodução.

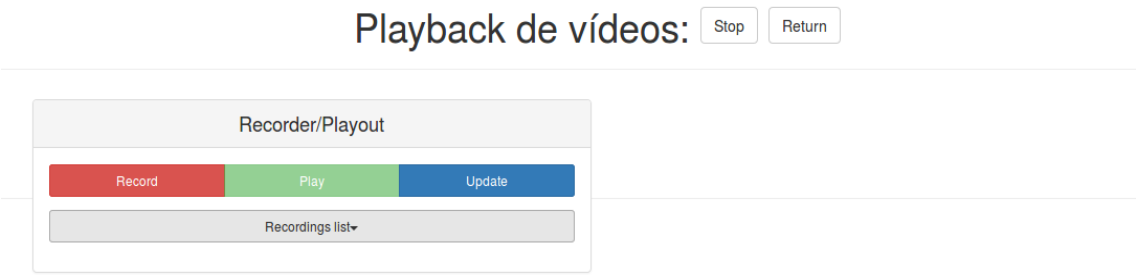

Figura 4.8: Record&Play implementado

A figura 4.8 mostra a página onde é possível utilizar a funcionalidade de playback. Esta implementação é baseada num exemplo disponibilizado pelos criadores do Janus que foi alterado para dar resposta às necessidades da plataforma.

Capítulo 5

Testes e Resultados

5.1 Testes

Findos os trabalhos de implementação, realizaram-se um conjunto de testes, descritos de seguida.

5.1.1 Testes de chamada

Foram realizados testes relativos à nova funcionalidade adicionada à plataforma relacionado com o estabelecimento de uma chamada entre dois utilizadores que se encontram em instâncias das salas imersivas instaladas em domínios diferentes. Foram simulados diferentes casos possíveis para confirmar que a solução funciona em qualquer destes casos. Num primeiro teste, foi realizada uma chamada utilizando um utilizador registado e controlando uma sala (condição minimamente necessária) e um outro utilizador *ad-hoc* que não precisa de registo mas tem acesso ao diretório de chamadas. A chamada foi realizada com sucesso. Um outro caso testado foi o estabelecimento de uma chamada entre dois utilizadores registados e cada um controlando uma sala em seu devido domínio. Também concluindo a chamada com sucesso.

5.1.2 Testes de robustez

O teste de robustez teve por objetivo por à prova o *script* criado com o intuito de monitorizar os serviços dos quais depende a plataforma para funcionar e, no limite, recuperar o estado ativo caso um destes sofra um *crash* durante uma chamada. Para dar mais credibilidade a este teste, foi simulado um *crash* no Janus durante uma chamada para verificar se este teria capacidade de resposta para este problema e, tal aconteceu, sendo que o *script* é executado a cada 15 segundos, onde se ocorrer um erro em algum dos serviços necessários, o *script* deteta o serviço em falta e procede ao *deploy* do mesmo.

5.2 Resultados

Os resultados conseguidos após realizar os testes nas condições descritas na secção anterior, serão aqui apresentados em forma de capturas de ecrã que demonstram o sucedido durante tais testes, a fim de servir como prova dos mesmos. De notar que algumas capturas são feitas do ecrã na sua totalidade pois é de interesse deixar visível a barra de endereços para poder identificar os domínios onde cada utilizador se encontra, especialmente nas chamadas para utilizadores em domínios diferentes.

5.2.1 Resultados do teste de conectividade

Este teste vai recorrer a dois utilizadores, "kuku" e "bodja", registados em instâncias diferentes e que pretendem realizar uma chamada entre si, com a finalidade de testar se é possível conectar dois utilizadores que não se encontram no mesmo domínio.

Na figura 5.1 a sala 5 está sendo controlada pelo utilizador "kuku".

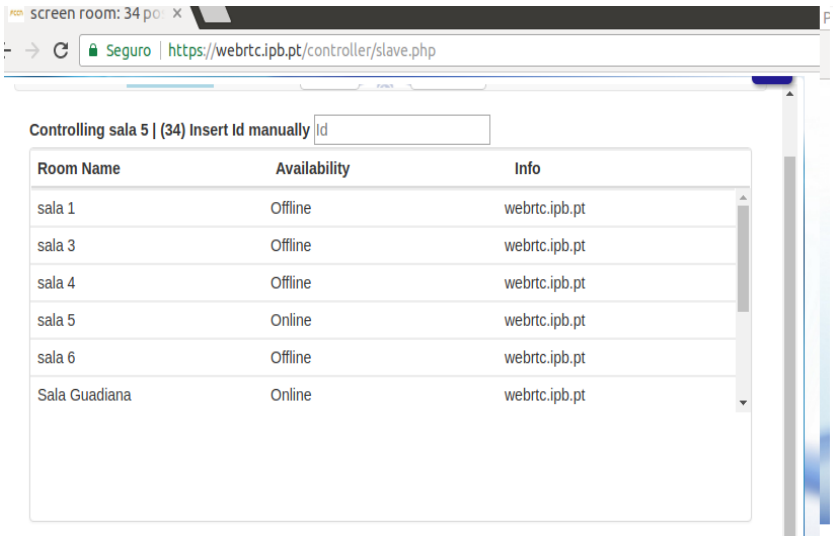

Figura 5.1: Utilizador "kuku" controlando a sala 5

Já na figura 5.2 a sala 1 que se encontra em outro domínio é controlada pelo utilizador "bodja".

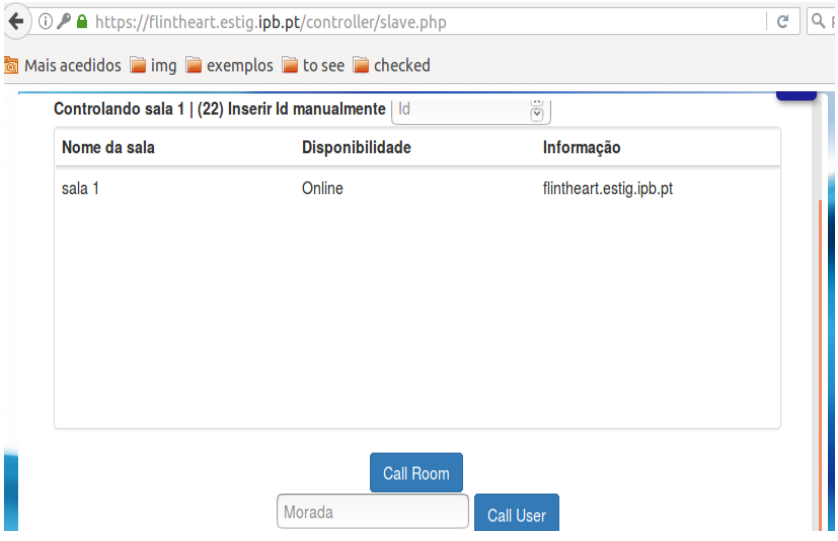

Figura 5.2: Utilizador "bodja" controlando a sala 1

A figura 5.3 pertence ao utilizador "bodja" que pretende ligar ao utilizador "kuku" que se encontra no domínio "webrtc.ipb.pt".

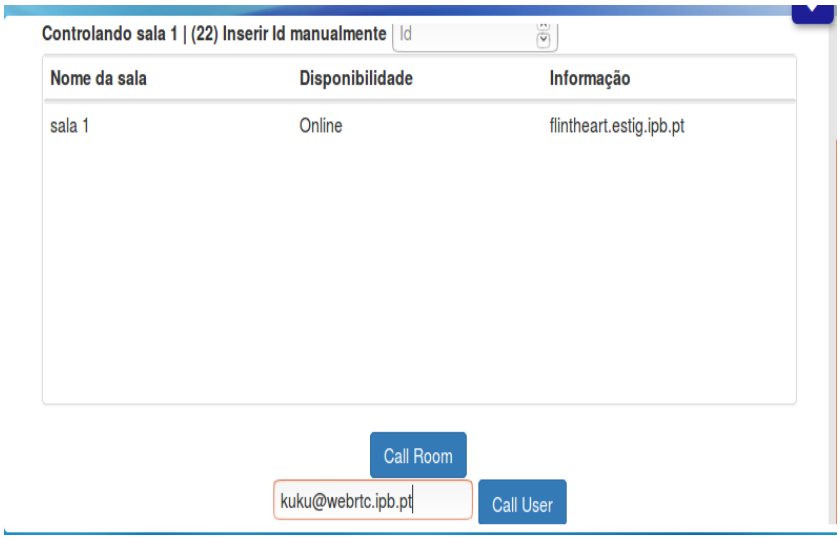

Figura 5.3: Utilizador "bodja" inserindo o full name de "kuku".

Segue-se a figura 5.4 que demonstra a migração do utilizador "bodja" para o servidor onde se encontra o utilizador que pretende ligar.

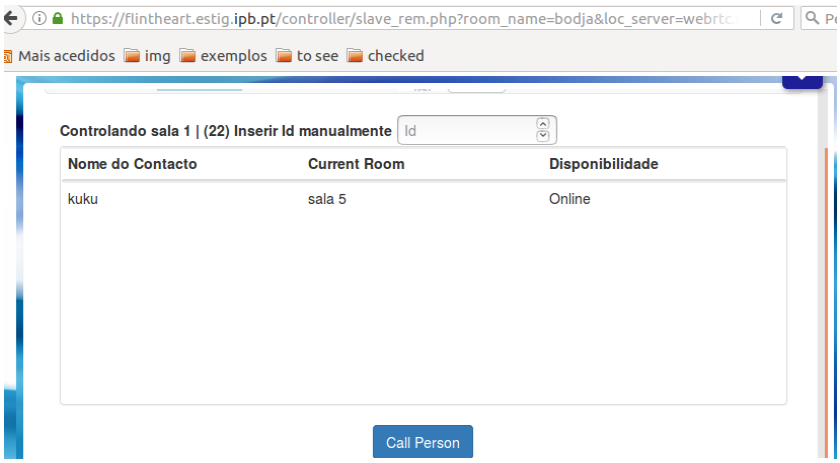

Figura 5.4: Utilizador "bodja" transferido para um diretório em "webrtc.ipb.pt" para realizar a chamada

Nas restantes figuras 5.5 e 5.6 são apresentados quando um utilizador consegue reunir todas as condições para ligar ao destino e procede ao pedido de chamada, sendo que a segunda figura representa o pedido que chega ao destino da chamada.

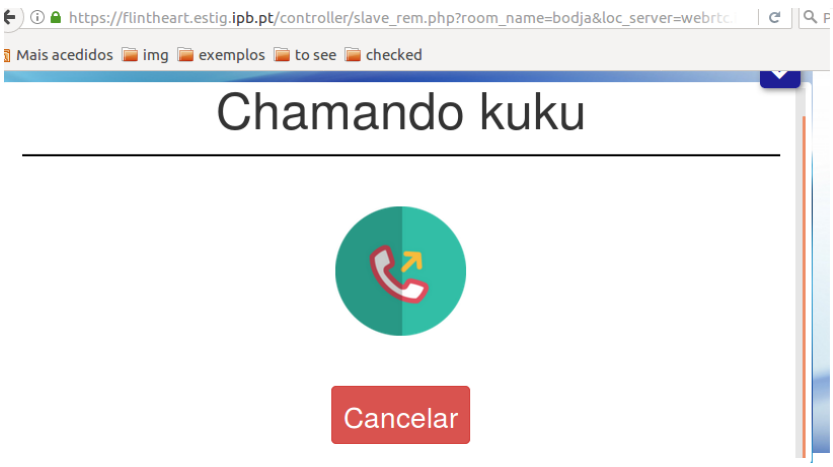

Figura 5.5: Utilizador "bodja" iniciando a chamada

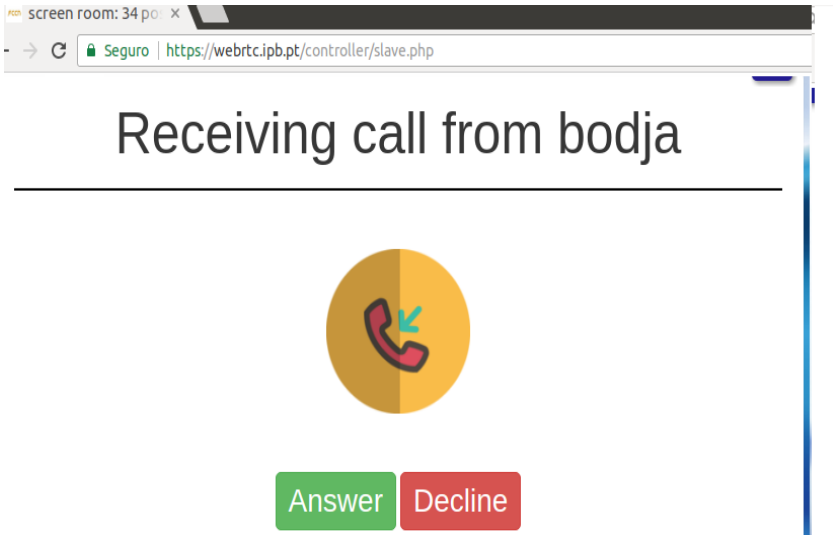

Figura 5.6: Utilizador "kuku" recebendo a proposta de chamada

5.2.2 Resultados do teste de robustez

Teste efetuado para ver se o *script* consegue dar resposta aos problemas que podem ocorrer.

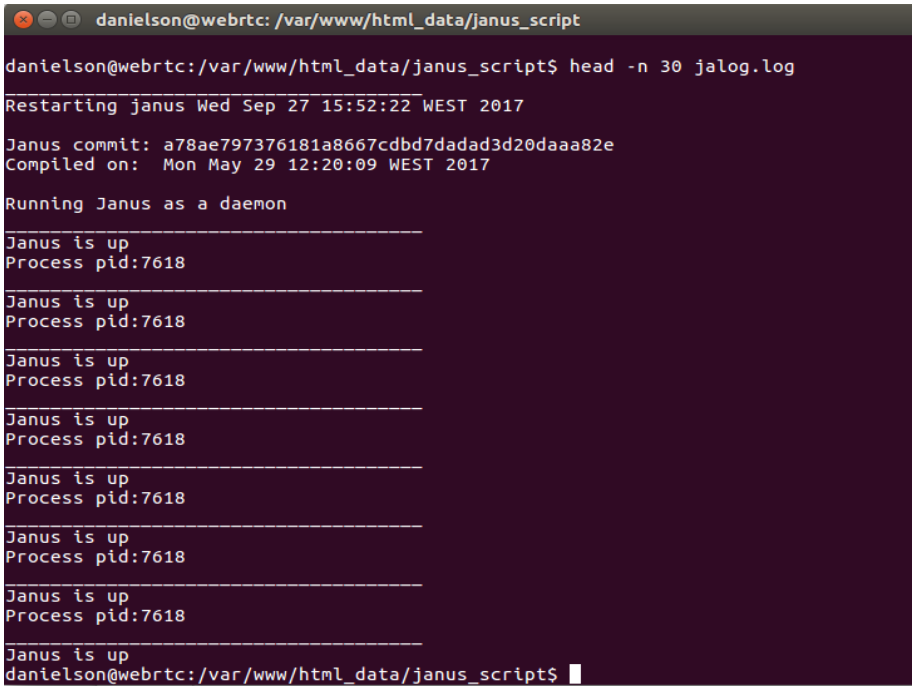

Figura 5.7: Log do script

Como é possível identificar na figura 5.7, as alterações de estado do Janus são monitorizadas e guardadas no ficheiro de *log* definido no *scrpit* para no caso de se pretender obter informação detalhada do funcionamento deste serviço.

Capítulo 6

Conclusões e trabalho futuro

6.1 Conclusão

Após as fases de estudo da plataforma e das tecnologias que compõe o WebRTC, foram feitos avanços em termos de implementação de funcionalidades na plataforma e melhoria na estabilidade da mesma. Desta forma, possível concluir com sucesso a adição de uma nova funcionalidade, as chamadas multidomínio, sendo agora possível realizar chamadas entre utilizadores registados em domínios diferentes. No que concerne à melhoria da estabilidade, foi desenvolvido um *bash script* com o intuito de monitorizar o estado dos serviços dos quais o WebRTC depende e, caso um destes falhe, o *script* trata de arrancar o mesmo.

Foi também realizado um trabalho de pesquisa de modo a identificar a melhor opção de implementação da funcionalidade de *playback* onde já foi possível identificar o necessário para fazer com que esta funcionalidade fique disponível. Sendo que esta já se encontra implementada e parcialmente funcional, sendo que não é ainda um processo totalmente automatizado.

6.2 Trabalho futuro

Como trabalho futuro relativamente a este projeto de salas imersivas, é necessário melhorar os processos de interação com a plataforma, visando uma melhor usabilidade por parte dos utilizadores da mesma. Por outras palavras, tornar a plataforma mais *user-friendly*, de modo a que todos os recursos que a mesma possui sejam facilmente acessíveis e simples de utilizar.

A plataforma possui a capacidade de gravar as chamadas, porém falta uma outra parte importante, a reprodução do conteúdo gravado. Esta funcionalidade está na sua fase final de implementação na data de escrita deste documento, pelo que não deve tardar a estar concluída.

A estabilidade dos componentes da mesma também é uma questão que levanta alguma preocupação, pois embora tenha sido encontrada uma solução para contornar o problema, seria de melhor utilidade descobrir o ponto crítico do problema e resolver em definitivo o mesmo.

Uma funcionalidade que também teria um impacto positivo na plataforma seria a implementação de um sistema de partilha de ficheiros, durante uma vídeo-conferência. Este objetivo pode ser conseguido dando uso aos *DataChannels* que foram previamente descritos nesta tese e que possibilitam a implementação de tal funcionalidade.

Bibliografia

[1] W3. (ago. de 2017). Webrtc 1.0: Real-time communication between browsers, endereço: `https://www.w3.org/TR/webrtc/`.

[2] G. G. Bernardo. (fev. de 2014). Webrtc beyond one-to-one communication, endereço: `https://webrtchacks.com/webrtc-beyond-one-one/`.

[3] A. Ramirez, «Webrtc and multi-party video», jun. de 2014.

[4] L. Byrd, «The unbearable lightness of webrtc signaling», 22 de ago. de 2013.

[5] U. C. L. Mark Handley, «[rfc4566]:session description protocol», rel. téc., jun. de 2006.

[6] S. Ludwig, «Jingle», rel. téc., mai. de 2016.

[7] A. B. Johnston e D. C. Burnett, *Webrtc: Apis and rtcweb protocols of the html5 real-time web*. USA: Digital Codex LLC, 2012, ISBN: 0985978805, 9780985978808.

[8] E. Marocco, «Signalling options for webrtc applications», 2 de set. de 2013.

[9] W. Ancheta, «Building a webrtc video chat application with peerjs», 7 de jun. de 2016.

[10] B. Castillo, «[rfc6455]websocket», rel. téc., nov. de 2013.

[11] D. Ristic, «Webrtc summit», em *WebRTC data channels*, 4 de fev. de 2014.

[12] S. Dutton, «Webrtc in the real world: Stun, turn and signaling», 4 de nov. de 2013.

[13] A. Prokop, «An introduction to webrtc signaling», 16 de jul. de 2014.

[14] P. Srisuresh, «[rfc2663] ip network address translator (nat) terminology and considerations», rel. téc., ago. de 1999.

[15] K. Egevang, «[rfc1631]the ip network address translator (nat)», rel. téc., mai. de 1994.

[16] J. Tyson, «How network address translation works», 2 de fev. de 2001.

[17] J. Rosenberg, «[rfc5389]session traversal utilities for nat (stun)», rel. téc., out. de 2008.

[18] P. Srisuresh, «[rfc5128] state of peer-to-peer (p2p) communication across network address translators (nats)», rel. téc., mar. de 2008.

[19] R. Mahy, «[rfc5766]traversal using relays around nat (turn)», rel. téc., abr. de 2010.

[20] J. Rosenberg, «[rfc5245]interactive connectivity establishment (ice)», rel. téc., abr. de 2010.

[21] E. Ivov, «Ice always tastes better when it trickles», 4 de dez. de 2013.

[22] L. Miniero, «What is a webrtc gateway anyway», 19 de abr. de 2014.

[23] L. Mineiro. (2014). Janus webrtc gateway, endereço: `https://janus.conf.meetecho.com/docs/index.html`.

[24] J. Liang. (mar. de 2016). Four exciting webrtc applications to watch in 2016, endereço: `https://www.onsip.com/blog/four-exciting-webrtc-applications-to-watch-in-2016`.

[25] WebRTCHacks. (2017). Webrtc developer tool vendor directory, endereço: `https://webrtchacks.com/vendor-directory/`.

[26] WebRTCWorld. (2017). Sites that uses or demo webrtc, endereço: `http://www.webrtcworld.com/webrtc-list.aspx`.

[27] F. FCCN. (2017). Fccn ao serviço do ensino e da ciência, endereço: `https://www.fccn.pt/`.

[28] A. Gulbrandsen, «[rfc2782]a dns rr for specifying the location of services (dns srv)», rel. téc., fev. de 2000.

[29] OnSIP, «Sip dns srv records - quick guide», fev. de 2016.

Apêndice A

Proposta Original do Projeto

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Master degree of Information Systems

2016/2017

Dissertation proposal

Title: Productize WebRTC Multi-screen, Multi-point Videoconference Solution

Supervisor: Nuno Rodrigues

Co-Supervisor: Rui Ribeiro (FCT|FCCN)

1 Goals

During the last 2 years, FCT|FCCN has developed a solution that provides an alternative to the traditional h.323 videoconference rooms. This WebRTC based solution, provides more functionality and is more flexible than current systems. The solution is deployed over standard hardware (of shelve equipment) and can be installed with any number of screens. It provides multi-screen, multi-point, HD video quality with recording. The platform also supports user and room profiles that can be accessed thru a mobile device.

The goal is to transform the current prototype into a full product. The room solution should be updated to allow easier usage, easier room setup, multi-domain calls and record playout. The central solution must be updated to provide high performance, high resiliency.

The final delivery will be the redistributable software package that supports a fully featured domain and a set of documentation with instructions and procedures to maintain the service.

2 Details

The current prototype version was implemented in two different stages by two undergraduate students:

1. Service setup: directory, users, room management, point to point call
2. Service improvement: multi-point, recording

On this third stage the goal is to make the final step towards the final product. This stage will require: multi-domain calls, record playback, better usability for final users and room managers, high performance and high resiliency.

The work should follow the following structure:

1. Technology review and implementation of dedicated service infrastructure. (20%)
2. Establish Quality Assurance mechanisms (service monitoring and resilience) (20%)
3. Usability tests over the prototype and study of improvement integration. (20%)
4. Implementation of all the improvements identified and integration of the “multi-domain” feature. (20%)
5. Write operation procedures for deployment and operation of the service (20%)

During the process, the student will increase his knowledge on: “Product Development”, UX, Usability, Service Management, HTML, CSS, Javascript (WebRTC, socket.io), NodeJS, PHP, MySQL and Janus Gateway.

Apêndice B

Outro(s) Apêndice(s)

Manual de instalação das Salas Imersivas Webrtc

SO:

O sistema operativo recomendado para a instalação das Salas Imersivas passa por uma distribuição Linux que seja minimamente robusto quer seja de 32 ou 64 bits. As instâncias que foram utilizadas para testes e desenvolvimento, foram instaladas em Debian 8.

Hardware:

Em termos de *hardware*, o recomendado passaria por uma máquina com pelo menos um processador dual core de 2,5GHz, 4GB de memória RAM, espaço em disco de 20GB e uma conexão ethernet de 100MB/s. Isto seria o suficiente para que a plataforma funcione normalmente em ambiente de testes.

Coturn:

- (Sudo) apt-get install coturn
- Alterar o ficheiro `/etc/turnserver.conf` de acordo com as instruções que se encontram descritas no mesmo.
- Arrancar serviço com o comando `turnserver [options]`

PeerJS:

- Descarregar o projeto do repositório fornecido pela FCCN.

- Criar um ficheiro `config.js` utilizando o exemplo `config.js.sample` como base
- Correr os comandos no terminal:
  - `npm install` para instalar dependências.
  - `bash apply_fixes.sh` para corrigir alguns pontos.
- Utilizar o Forever(ferramenta da linha de comandos) para correr o script de forma ininterrupta:
  - Instalar forever com o comando npm `install forever`
  - `forever [actions] [options] server.js`

Janus:

- Para a instalação do janus são necessárias algumas packages adicionais mas só serão aqui apresentadas as fulcrais para o funcionamento das salas imersivas:
  - Jansson (dev)
  - libnice (dev)
  - OpenSSL (dev)
  - libwebsockets
  - libcurl (dev)
  - glib (dev) 2.0
  - pkg-config 0.28
  - libtool 2.4.6.40
  - autoconf 2.69
  - automake 1.15

- Após instalar as dependências acima, compilar o janus:
  - Descarregar o janus do repositório `https://github.com/meetecho/janus-gateway.git`
  - `cd janus-gateway`
  - `sh autogen.sh`
  - `./configure --prefix=/opt/janus`
  - `make`
  - `make install`
  - `make configs`

- Os ficheiros de configuração que precisam ser alterados encontram-se em `/opt/janus/etc/janus/`

WebRTC Immersive:

- Após instalar os outros componentes, descarregar o projeto “salas imersivas” do repositório também fornecido pela FCCN.
- Configurar o apache para arrancar o index page a partir de `/webrtc-immersive/html/` .
- Alterar `/webrtc-immersive/libs/config.php` de acordo com suas configurações locais. Importante configurar os serviços janus, nodejs e coturn nas mesmas portas que no ficheiro config.php .
- É preferivel utilizar o coturn como STUN/TURN server para ter mais controlo sobre a aplicação e ter acesso a mais informação já que os logs serão gerados localmente.
- Dentro do diretório `/webrtc-immersive/html` criar links para `/webrtc-immersive/assets` e também para `/webrtc-immersive/controller`.
- Utilizar o script que se encontra em `janus-script/` para manter o janus ativo, com o seguinte comando : `sudo nohup bash januscrpt2.sh &`
- Chamadas para contatos através do “dashboard” ou do “slave” só funcionam para targets que estejam existam e estejam online, caso contrário falha devido às verificações de dominio, serviço, utilizador e state.
- Em `html/registered_screen.php` alterar o valor da variável “$key“ na linha 62, com a chave do google url shortener para a sua respetiva chave.
- É importante manter todos os serviços necessários ao funcionamento das salas imersivas a serem executados como daemons.