Mestrado em Engenharia Informática
Dissertação/Estágio 2015/2015
Relatório Final

WebRTC as a Service

Pedro Henrique Lopes Martins
phlopes@student.dei.uc.pt

Orientador da WIT Software

Tiago Lopes Leitão
tiago.leitao@wit-software.com

Orientador do DEI

Carlos Lisboa Bento
bento@dei.uc.pt

1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática
Dissertação/Estágio 2015/2015
Relatório Final

WebRTC as a Service

Pedro Henrique Lopes Martins
phlopes@student.dei.uc.pt

Orientador da WIT Software
Tiago Lopes Leitão
tiago.leitao@wit-software.com

Orientador do DEI
Carlos Lisboa Bento
bento@dei.uc.pt

Júri Arguente
Edmundo Monteiro
edmundo@dei.uc.pt

Júri Vogal
Tiago Baptista
baptista@dei.uc.pt

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Atualmente, com a constante evolução do mercado de dispositivos móveis, o mercado dos serviços de comunicação começou a crescer e os tradicionais serviços de comunicação começaram a perder clientes. Os novos serviços aproveitam as vantagens da comunicação pela internet, oferecendo assim um serviço melhor que os serviços de telecomunicação.

Muitos de destes serviços apenas permitem a comunicação com clientes que usem o mesmo serviço, o que pode reduzir o ser interesse, e para criar um novo produto é necessário criar preparar infraestruturas próprias para lidar com a comunicação dos clientes, o que aumenta a complexidade do desenvolvimento e manutenção de soluções de comunicação.

O WebRTC é uma tecnologia em crescimento que tem vindo a ser cada vez mais utilizada no desenvolvimento de novas soluções e que tem vindo a ser integrada nas soluções existentes. Esta visa trazer capacidade de comunicação em tempo real para os *browsers* e sem a necessidade de instalar *plugins*, atuando como uma camada de compatibilidade entre os serviços de comunicação, reduzindo a complexidade do desenvolvimento e migração de soluções de comunicação.

A WIT Software disponibiliza uma solução que faz uso do WebRTC para facilitar o desenvolvimento de aplicações e serviços de comunicação: a WIT WebRTC Gateway e um SDK. Esta *gateway* disponibiliza todas as funcionalidades necessárias para comunicar por WebRTC e integra-se facilmente nas infraestruturas de comunicação dos clientes, permitindo a intercomunicação entre serviços de telecomunicação tradicionais e utilizadores WebRTC.

O objetivo deste projeto é a criação de uma prova de conceito de uma solução que agilize a distribuição e comercialização da WIT WebRTC Gateway, disponibilizando uma plataforma onde os clientes podem subscrever o serviço e descarregar o SDK, e disponibilizando múltiplas instâncias da *gateway* às quais os clientes subscritores do serviço têm acesso para criar soluções de comunicação. O serviço deve permitir aos clientes descarregar o SDK configurado com a definições do servidor SIP dos cliente, e gerar chaves de acesso às *gateways*.

Palavras-chave: ”comunicação”, ”gateway”, ”SDK”, ”SIP”, ”WebRTC”

Abstract

Nowadays, with the constant evolution of the mobile devices market, the market of communication services started to grow and the traditional telecommunications services started to lose clients. The new services take profit of the advantages offered by the communication via internet (reduced latency and increased quality due to larger bandwidth), offering a service that is better than the traditional communication services.

Many of these services only allow communication with clients using the same service, which may reduce their interest, and to create new product it's necessary to build specialized infrastructures capable of dealing with clients' communications, which increases the complexity in the development and maintenance of communication services.

WebRTC is a growing technology that is being adopted by many new and existing communication solutions. This technology brings real-time communication capabilities between browsers and without the need of plugins, acting as a compatibility layer for communication services and reducing the complexity in the development and maintenance of communication services.

WIT Software owns a solution that makes use of WebRTC to allow to create communication applications and services: WIT WebRTC Gateway and an SDK. This gateway provides all necessary functionalities to communicate via WebRTC and easily integrates with clients' communication infrastructures, allowing intercommunication between traditional telecom services and WebRTC peers.

The goal of this project is to create a proof of concept of a solution that speeds up the distribution and commercialization of the provided gateway service, by creating a platform where clients can subscribe the service and download the SDK, and by providing multiple instances of the gateway that clients can access so they can build communication solutions. The service should allow clients to download the SDK with the configuration from their own SIP servers, and to generate access keys.

Keywords: ”communication”, ”gateway”, ”SDK”, ”SIP”, ”WebRTC”

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CSS	Cascading Style Sheets
DBMS	Database Management System
DDoS	Distributed Denial of Service
DoD	Definition of Done
DoS	Denial of Service
GB	Giga Byte
GUI	Graphics User Interface
HTML	Hypertext Markup Language
I/O	Input/Output
IaaS	Infrastructure as a Service
ICE	Interactive Connectivity Establishment
IP	Internet Protocol
IP-PBX	SIP-based Private Branch Exchange
ITSP	Internet Telephony Service Provider
JSON	JavaScript Object Notation
MMS	Multimedia Messaging Service
NAT	Network Address Translator
P2P	Peer-to-peer
POC	Proof of Concept
PSTN	Public Switched Telephone Network
REST	Representational State Transfer
SBC	Session Border Controller
SDK	Software Development Kit

SIP	Session Initiation Protocol
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TURN	Traversal Using Relays around NAT
VoIP	Voice over Internet Protocol
WebRTC	Web Real-Time Communication

Glossário

API	Conjunto de métodos disponibilizados por um serviço para que outros serviços ou utilizadores possam efetuar ações sobre o serviço
Backend	Componente de administração do serviço
Backoffice	Componente de administração não visível para os utilizadores finais do serviço
Browser	Navegador de *Internet*
Capacidade de uma chave	Número de chamadas concorrentes autorizadas para uma chave
Chave de serviço	Dados de do serviço subscrito, com informação acerca da validade, capacidade e *key secret*
Framework	Biblioteca de *software* com funcionalidades que ajudam no desenvolvimento de *software*
Frontend	Interface que permite a ao cliente comunicar com o *backend*
Gateway	Componente que atua como interface para protocolos para permitir a troca de informação entre componentes que usem diferentes protocolos
I/O	Indica a transição (entrada e saída) de dados entre um sistema e um componente ou sistema externo
Key secret	*Token* usado juntamente com o SDK para fazer a autenticação e validação dos pedidos na *gateway*

Multi-tenancy	Capacidade de um *software* disponibilizar e distribuir partes de uma instância a diferentes grupos de utilizadores (cada um com diferente informação, configuração, gestão e funcionalidades)
Proxy	Servidor que atua como intermediário no pedido de recursos feitos por cliente a um serviço
Scrum	Metodologia ágil de desenvolvimento de *software*
Serviço de sinalização	Serviço que permite encontrar clientes na *internet* e trocar as suas informações para poderem comunicar
Software as a Service	Tipo de serviço de computação na cloud em que é disponibilizado acesso a uma aplicação produto
Sprint	Na metodologia Scrum, é o período de tempo para o desenvolvimento de cada iteração do projeto

Capítulo 1

Introdução

O presente relatório, juntamente com os seus anexos, tem como objetivo documentar o trabalho desenvolvido ao longo do estágio na empresa WIT Software, no âmbito da unidade curricular Dissertação/Estágio em Engenharia de Software do Mestrado em Engenharia Informática na Universidade de Coimbra.

O estágio foi orientado pelo pelo Professor Carlos Bento, como orientador pelo Departamento de Engenharia Informática, e pelo Engenheiro Tiago Leitão, como orientador pela WIT Software.

A WIT Software é uma empresa especializada no desenvolvimento de soluções para operadores móveis e empresas de telecomunicação. A empresa tem clientes em vários países, sendo que alguns são líderes de mercado na área de telecomunicações, nomeadamente a Deutsche Telekom, Orange, Telefónica, TeliaSonera, Unitel e Vodafone.

1.1 Contexto

Recentemente, o aparecimento de serviços de comunicação sob a forma de aplicações tem vindo a alterar a forma como os serviços de comunicação operam e são monetizados. As empresas que disponibilizam serviços de telecomunicação tradicionais começaram a perder clientes e começaram a disponibilizar um serviço de acesso à internet para reaproveitar e rentabilizar as suas infraestruturas.

Muitas empresas que necessitam de serviços de comunicação ainda procuram soluções mais económicas, no entanto a maioria não tem os recursos necessários para manter infraestruturas com capacidade para lidar com as suas necessidades, e as soluções existentes nem sempre se adaptam às diferentes necessidades das empresas. Outro grande problema é a falta de compatibilidade entre soluções, que leva a que a migração de soluções se torne um processo moroso e dispendioso.

1.2 WebRTC

O WebRTC é uma tecnologia em crescimento que tem vindo a ser cada vez mais utilizada no desenvolvimento de novas soluções e que tem vindo a ser integrada nas soluções existentes. Esta tecnologia tem potencial para mudar a forma como comunicamos, uma vez que visa trazer capacidade de comunicação em tempo real para os browsers e sem a necessidade de instalar plugins, reduzindo a complexidade do desenvolvimento de aplicações de comunicação. Além disso, já inclui funcionalidades importantes como a confiabilidade e a segurança. Esta tecnologia é apresentada de forma mais aprofundada no **Capítulo 2.2.2** .

A WIT Software disponibiliza uma solução que faz uso do WebRTC para facilitar o desenvolvimento de aplicações e serviços de comunicação: a WIT WebRTC Gateway. Esta gateway disponibiliza todas as funcionalidades necessárias para comunicar por WebRTC e integra-se facilmente nas infraestruturas de comunicação dos clientes, permitindo a intercomunicação entre serviços de telecomunicação tradicionais e utilizadores WebRTC. A solução inclui ainda um SDK pronto a comunicar com a gateway e que facilita o desenvolvimento de soluções de comunicação.

A solução atual requer que a gateway seja instalada nas infraestruturas dos clientes (sejam infraestruturas privadas ou um serviço de cloud), e requer que os clientes entrem em contacto com a empresa para negociar o contrato e configurar o serviço para o integrar nas infraestruturas de comunicações dos clientes.

1.3 Objetivos

O objetivo deste estágio é a criação de uma prova de conceito de uma solução que agilize a distribuição e comercialização da WIT WebRTC Gateway, oferecendo o SDK e disponibilizando um serviço de cloud SaaS (Software as a Service), constituído por múltiplas instâncias da gateway às quais os clientes subscritores do serviço têm acesso para criar soluções de comunicação. Isto tudo num serviço que dispensa intervenção por parte da empresa.

O público alvo do do serviço a desenvolver são empresas de desenvolvimento de *software* que tenham acesso a um servidor SIP e que pretendam criar soluções de comunicação que integrem WebRTC. Na plataforma *web* vão poder configurar o SDK com os dados do seu servidor SIP para poder estabelecer a comunicação entre as gateways da WIT e a sua infraestrutura de comunicação. O protótipo deve ainda impor limites sobre a utilização do serviço, nomeadamente a validade do serviço adquirido e número de chamadas concorrentes.

Para cumprir o objetivo é necessário criar um protótipo com cinco componentes principais:

- **Criação de uma plataforma de gestão**: para dispensar a intervenção da empresa na comercialização do serviço é necessário disponibilizar aos clientes uma plataforma onde estes se possam registar, adquirir o SDK e pagar pelo serviço de uma forma fácil. Para gerir as contas existentes deve ainda haver uma versão da plataforma para o administrador da empresa.
- **Criação de um backoffice de gestão**: para gerir as contas dos clientes e o serviço, é necessária a criação de um backoffice que disponibilize o acesso à plataforma e que faça a persistência da informação do serviço. É ainda necessário que esta disponibilize endpoints REST para a comunicação entre a plataforma e o backoffice (para efetuar ações sobre a gestão do serviço), e enpoints REST para as gateways terem acesso à informação do serviço.
- **Integração de um serviço de pagamentos**: a plataforma necessita de disponibilizar um meio de pagamentos para que os clientes tenham acesso ao serviço. A componente de pagamentos de uma plataform apresenta uma alta complexidade, uma vez que é um ponto crítico do ponto

de vista da segurança e confiabilidade. Para agilizar o desenvolvimento da plataforma e para oferecer um meio de pagamentos com que o cliente se sinta mais seguro, esta componente vai ser feita recorrendo a um serviço externo de pagamentos que deve ser integrado com o backoffice e a plataforma.

- **Centralização da gestão do serviço**: o acesso às funcionalidade da gateways só é permitido após a autenticação, que é feita utilizando o *key secret* de uma chave de serviço (API Key), chave essa que tem restrições previamente acordadas (como a validade e o número de chamadas concorrentes). A versão atual da gateway não tem implementada a contagem de chamadas concorrentes (não é imposto um limite aos clientes uma vez que a gateway é instalada nas suas infraestruturas). Além disso não permite ter uma infraestrutura com múltiplas gateways com a gestão das chaves do serviço de forma centralizada, para poder balancear a carga do serviço mas mantendo as restrições de utilização. Nesta prova de conceito pretende-se alterar as gateways para que passem a obter a informação das chaves e que façam a validação das restrições comunicando com o backoffice, e pretende-se que façam a contagem da utilização local das chaves e que sincronizem essa informação com o backoffice para que este mantenha a contagem da utilização das chaves por todas as gateways para gerir as restrições dos serviços acordados.
- **Integração de um serviço de balanceamento de carga geográfico**: num serviço de comunicação é muito importante conseguir oferecer a menor latência possível para que a comunicação pareça mesmo em tempo real. Um balanceador de carga geográfico permite distribuir os pedidos dos clientes, pelas instâncias de um serviço, com base na sua localização geográfica, o que permite que lhes sejam atribuídas as instâncias que se encontram mais perto deles, reduzindo a latência na comunicação. Nesta prova de conceito pretende-se integrar um serviço de balanceamento de carga geográfico para reduzir a latência na comunicação entre os cliente e as gateways.

1.4 Estrutura do documento

Esta secção conclui o primeiro capítulo e tem como objetivo descrever de um modo geral o conteúdo apresentado nos próximos capítulos.

No segundo capítulo é apresentada uma análise feita aos competidores e um estudo das tecnologias e serviços a considerar para o desenvolvimento do projeto.

No terceiro capítulo é apresentada a metodologia de desenvolvimento, o planeamento e os riscos.

No quarto capítulo é dividido em dois grupos: o primeiro descreve a arquitetura do sistema constituído pela camada de gestão e pela camada de comunicação, o segundo grupo descreve a arquitetura da plataforma web.

No quinto capítulo é apresentada a lista de testes funcionais realizados.

No sexto e último capítulo é feita uma reflexão sobre trabalho desenvolvido e são feitas algumas sugestões para um trabalho futuro.

Este relatório é complementado por um conjunto de anexos com documentação confidencial acerca do desenvolvimento e do protótipo desenvolvido.

Capítulo 2

Estado da Arte

Neste capítulo é apresentada a análise feita aos competidores, com descrição dos seus produtos e respetivas funcionalidades, de forma a encontrar os requisitos mais importantes para a aplicação pretendida. Na segunda parte deste capítulo são apresentadas as tecnologias escolhidas para usar no desenvolvimento da aplicação, com base no tipo aplicação e nas tecnologias utilizadas em aplicações semelhantes.

2.1 Competidores

A análise de competidores permite encontrar as funcionalidades mais importantes para um tipo de aplicação, com base nas funcionalidades mais comuns entre os produtos dos competidores. Com base nas funcionalidades menos comuns é possível identificar funcionalidades menos importantes ou funcionalidades que permitam que um produto se destaque entre os seus competidores.

Os competidores dividem-se em dois grupos: diretos, que são os que oferecem um serviço na *cloud*, e indiretos, que são os que oferecem um sistema que os clientes podem instalar na sua rede.

2.1.1 Competidores diretos

Os competidores diretos são aqueles que fornecem uma *gateway* na *cloud* com suporte a SIP e capaz de receber e redirecionar chamadas WebRTC e VoIP.

Este é o serviço mais recomendado para pequenas empresas, uma vez que não há necessidade de configurações avançadas nos vários módulos do sistema e fornecem um serviço que satisfaz as necessidades dessas empresas e que é capaz de escalar de quando as necessidades dessas empresas mudam.

Twilio

Twilio é uma empresa de comunicações na cloud que oferece um serviço de comunicação compatível com WebRTC, VoIP e sistemas de telecomunicação tradicionais (PSTN). Permite comunicação de áudio e vídeo e já disponibiliza servidores STUN e TURN. Disponibilizam também um *kit* de desenvolvimento (SDK) que facilita o desenvolvimento de aplicações e serviços de comunicação que façam uso das suas infraestruturas para tratar de partes mais complicadas como a sinalização e as filas de espera. Os pagamentos são feitos adicionando fundos à conta, que pode ser feito através de um cartão de crédito ou PayPal.

Figura 2.1: Logo da Twilio

As principais funcionalidades do serviço da Twilio são:

- Gravação de chamadas
- Notificações em tempo real
- Fácil criação de conferências de áudio para clientes VoIP, clientes PSTN e *endpoints* SIP.
- Filas de chamadas
- Servidor STUN gratuito
- Registos de utilização
- *Multi-tenancy* (para controlo de utilização por grupos e estatísticas)
- Painéis para visualização da utilização do serviço em tempo-real
- Suporte a SMS e MMS
- Compatibilidade com múltiplos fornecedores de telecomunicação
- Suporte a chamadas de emergência
- Optimizado para dispositivos móveis e computadores

Plivo

O serviço da **Plivo** é uma plataforma que permite fazer chamadas de voz e enviar SMS para todos os países. Este serviço fornece suporte nativo a SIP e disponibiliza um SDK WebRTC que facilita a comunicação entre *browsers* e PSTN sem restrições de países. Também disponibiliza números de telefone internacionais para compra, permitindo reduzir os custos para o utilizador final. Os pagamentos são feitos adicionando fundos à conta, sendo isto feito apenas através de um cartão de crédito.

Figura 2.2: Logo da Plivo

As principais funcionalidades do serviço da Plivo são:

- Chamadas de voz e SMS para todos os países
- Suporte nativo a SIP
- Suporte de *unicode* para todas as línguas
- Identificador de envio dinâmico
- SDK para dispositivos móveis e para aplicações *web*
- Suporte a conversão de texto para voz
- Gravação e filas de chamadas
- Notificações em tempo real

SPiDR

SPiDR WebRTC Gateway é um produto da **Genband**, uma solução que disponibiliza comunicação entre serviços de comunicação pela *internet* (como redes VoIP e WebRTC). Pode ser instalada na rede do cliente, numa *cloud* privada ou na própria *cloud* da Genband. A este serviço ainda faltam muitas funcionalidades (como serviços de mensagens, livro de contactos, configuração e registos de chamadas), no entanto estas podem implementadas e adicionadas ao serviço através do SDK disponibilizado.

Figura 2.3: Logo do SPiDR

As principais funcionalidades do SPiDR são:

- *Multi-tenancy*
- API REST extensível
- Instalável na rede do cliente, numa *cloud* privada ou no serviço de *cloud* da Genband
- Disponibiliza uma aplicação *web* pronta a aceder ao serviço através de qualquer dispositivo com acesso à *internet*

AhoyRTC

A **AhoyRTC** disponibiliza vários serviços de comunicação construídos com foco no WebRTC (e compatíveis com outros serviços de comunicação, como PSTN e VoIP), serviços que vão desde serviços de chat por voz a serviços de videoconferência.

Figura 2.4: Logo da AhoyRTC

Os serviços disponibilizados são:

- **AhoyRTC Gateway**: utiliza SIP para a sinalização e funciona como uma ponte de comunicação entre clientes WebRTC e PSTN
- **AhoyRTC Cloud**: trata do VoIP e permite criar aplicações *frontend* para comunicação com a ajuda da API disponibilizada
- **AhoyConference**: é um servidor que pode ser instalado na rede do cliente ou numa *cloud* privada e que permite criar facilmente videoconferências utilizando WebRTC

As principais funcionalidades dos serviços da AhoyRTC são:

- Sinalização para clientes WebRTC e PSTN
- Comunicação entre clientes WebRTC e PSTN
- Disponibiliza um balanceador de carga
- Servidor de videoconferências na rede do cliente ou *cloud* privada
- Fácil configuração de conferências

2.1.2 Competidores indiretos

Os competidores indiretos são aqueles que disponibilizam produtos ou serviços de comunicação compatíveis com WebRTC e VoIP sob a forma de uma *gateway* que tem obrigatoriamente de ser instalada e configurada num servidor privado (seja na rede do cliente ou num serviço de *cloud*). Este é um serviço que pode ser interessante para empresas grandes, mas no caso de empresas ou clientes de pequena dimensão que procuram este serviço podem perder o interesse, seja por não possuírem os recursos necessários para suportar este serviço ou por não pretenderem um serviço tão complexo.

Frafos

O produto da **Frafos** é a **ABC WebRTC Gateway**, uma solução que disponibiliza uma *gateway* com funcionalidades de WebRTC e SBC (permite fazer sinalização, configuração de chamadas e transmissão de multimédia numa rede de comunicação). Esta *gateway* pode ser instalada na rede do cliente ou num serviço de *cloud*, e pode ser integrado nas infraestruturas de comunicação do cliente, ligando-se ou substituindo o seu SBC. A Frafos disponibiliza uma versão de testes da sua *gateway*, mas para adquirir o serviço é necessário entrar em contacto com a sua equipa de vendas.

Figura 2.5: Logo da Frafos

São oferecidos três planos de migração:

- **Substituição**: o SBC do cliente é substituído pela ABC WebRTC Gateway (uma vez que também disponibiliza essa funcionalidade). Esta plataforma vai lidar com os serviços SIP e WebRTC.
- **Extensão**: a ABC WebRTC Gateway é instalada em paralelo com o SBC do cliente. Chamadas VoIP são geridas pelo servidor VoIP so cliente, e as chamadas WebRTC são geridas pela ABC WebRTC Gateway e reencaminhadas para o SBC.
- ***Cloud***: a ABC WebRTC Gateway é instalada um serviço *cloud*. As chamadas WebRTC são geridas para ABC WebRTC Gateway e são reencaminhadas para as infraestruturas do cliente.

Como medidas de segurança são oferecidas as seguintes funcionalidades: limitação da taxa de pedidos (para proteger contra ataques DoS) e ocultação da topologia da rede (para que qualquer informação acerca da estrutura interna seja anonimizada antes de ser reencaminhada para o exterior).

As principais funcionalidades da *gateway* são:

- Manipulação e controlo de sessões SIP
- Gestão através de uma interface gráfica
- Monitorização do sistema e da rede
- Medidas de segurança
- Alta disponibilidade
- Interfaces abertas (REST)

Sonus

A **Sonus** fornece um produto que permite a comunicação entre clientes WebRTC e SIP, bem como sinalização para cliente WebRTC. Este produto disponibiliza um um sistema de gestão centralizada para a *gateway* WebRTC e SBC, incluindo funcionalidades de monitorização, estatísticas e resolução de problemas. O SDK disponibilizado dá suporte a clientes com *browsers* e aplicações móveis não compatíveis com WebRTC. o Serviço também suporta *multi-tenancy*, permitindo a gestão e estatísticas para grupos.

Figura 2.6: Logo da Sonus

As medidas de segurança disponibilizadas são: ocultação da topologia, mitigação de ataques DoS e DDoS e proteção contra fraudes e roubo do serviço.

As principais funcionalidades são:

- Compatível com clientes SIP, PSTN e WebRTC
- Suporte através de um SDK
- Medidas de segurança
- *Multi-tenancy*

- Facilmente escalável
- Gestão centralizada para a *gateway* WebRTC e SBC
- Alta disponibilidade

Oracle Communications WebRTC Session Controller

Oracle Communications WebRTC Session Controller é uma solução da **Oracle** que agiliza o desenvolvimento e instalação de aplicações e serviços WebRTC. Inclui um serviço de sinalização para clientes SIP e *web*, um serviço de multimédia que ajuda no tratamento de multimédia transmitido por clientes WebRTC e SIP (incluindo criptografia de dados e codificação e descodificação de multimédia), e um SDK para extender as funcionalidades do serviço e ajudar no desenvolvimento de aplicações RTC, uma vez que inclui métodos que facilitam a autenticação de clientes, gestão de sessões e controlo de ligações.

Figura 2.7: Logo da Oracle

As principais funcionalidades são:

- Serviço de sinalização
- Transmissão de multimédia entre WebRTC e SIP
- Extensível através do SDK disponibilizado
- Altamente escalável
- Criptografia ao nível da camada de rede
- Codificação e descodificação de multimédia
- Protecção avançada contra ataques DoS e DDoS

2.1.3 Comparação de funcionalidades

As seguintes funcionalidades a ser comparadas nos diferentes serviços foram escolhidas selecionando as funcionalidades mais importantes que são fornecidas pelos competidores (como a compatibilidade de serviços de comunicação e funcionalidades das plataformas) e o tipo de serviço prestado (instalação ou serviço *cloud*).

Funcionalidade	**Twilio**	**Plivo**	**SPiDR**	**AhoyRTC**	**Frafos**	**Sonus**	**Oracle**
Instalável nas infraestruturas dos clientes	●	●	●	●	●	●	●
Instalável num serviço cloud	●	●	●	●	●	●	●
Disponibiliza o próprio serviço de cloud	●	●	●	●	●	●	●
Integrável com as soluções de telecomunicação dos clientes	●	●	●	●	●	●	●
Suporta gravação de chamadas	●	●	●	●	●	●	●
Compatível com SIP	●	●	●	●	●	●	●
Compatível com VoIP	●	●	●	●	●	●	●
Compatível com PSTN	●	●	●	●	●	●	●
Compatível com WebRTC	●	●	●	●	●	●	●
Compatível com VoLTE	●	●	●	●	●	●	●
Suporta chamadas de vídeo	●	●	●	●	●	●	●
Disponibiliza um serviço de sinalização pronto a utilizar	●	●	●	●	●	●	●
Compatível com outros serviços de sinalização	●	●	●	●	●	●	●
Suporta filas de chamadas em espera	●	●	●	●	●	●	●
Suporta SMS e MMS	●	●	●	●	●	●	●
Disponibiliza SDK	●	●	●	●	●	●	●
Suporta *browsers*	●	●	●	●	●	●	●
Suporta aplicações para dispositivos móveis	●	●	●	●	●	●	●
Permite a gestão através de uma interface gráfica	●	●	●	●	●	●	●
Disponibiliza monitorização do sistema	●	●	●	●	●	●	●
Disponibiliza notificações em tempo real	●	●	●	●	●	●	●
Permite uma fácil gestão de API Keys	●	●	●	●	●	●	●
Disponibiliza registos de sistema	●	●	●	●	●	●	●
Disponibiliza pagamentos fáceis na plataforma	●	●	●	●	●	●	●
Disponibiliza uma versão de testes	●	●	●	●	●	●	●
Qualidade da informação disponível na página de divulgação	9/10	6/10	4/10	3/10	2/10	5/10	4/10

Tabela 2.1: Comparação das funcionalidades dos serviços

Após a análise dos competidores, é possível verificar a maioria dos competidores procura oferecer uma *gateway* que permite aos clientes WebRTC comunicarem entre si, mas nem todos oferecem um serviço de sinalização que permita a sua identificação na rede, ficando esta parte por ter de ser implementada pelo cliente.

A maioria dos competidores não oferece um serviço compatível com outros serviços de comunicação (como PSTN e VoLTE), e alguns não disponibilizam as interfaces necessárias para integrar esses serviços. Em vez disso, disponibilizam um SDK e os clientes é que têm de criar aplicações que façam a ponte entre o serviço fornecido e os outros serviços de comunicação, ou então devem modificar os seus serviços para que tornem compatíveis. Isto trás problemas quando um cliente decide mudar do serviço de um competidor para o serviço de outro competidor, uma vez que tem que voltar a tratar do processo de integração.

Um dos problemas encontrados na maioria das soluções é o facto de não possuírem uma plataforma onde seja possível adquirir facilmente acesso ao serviço. Em vez disso, a maioria das soluções requer que o cliente entre em contacto com o fornecedor do serviço para negociar o acesso e os seus custos.

2.2 Tecnologias

A escolha das tecnologias tem um grande impacto no desenvolvimento e no resultado de um projeto. A escolha de uma boa *framework* agiliza o desenvolvimento de um projeto uma vez que já estão disponíveis certos módulos ou funcionalidades que já estão implementados e testados, reduzindo assim o tempo de desenvolvimento. A escolha de uma boa tecnologia ou serviço vai ter impacto no desempenho do resultado do projeto e na sua manutenção.

É por isso importante escolher as tecnologias que mais se adequam a cada caso, e é importante ter em consideração o tipo de projeto, o público alvo e a segurança.

Para este projeto pretende-se construir uma aplicação *web* que permite aos clientes e administrador gerir o serviço, e um aplicação servidor que vai receber os pedidos desses utilizadores e os pedidos das *gateways*. É por isso importante escolher uma boa *framework* para o *frontend*, uma *framework* para o *backend*, um meio para a persistência de dados e os serviços a integrar para

que a aplicação cumpra os requisitos (serviço de pagamentos e balanceamento geográfico).

2.2.1 WebSocket

WebSocket é um protocolo que permite comunicação *full-duplex* (sistema ponto a ponto composto por duas entidades ligadas que podem comunicar em ambas as direções) utilizando uma única ligação TCP. Isto significa que, ao contrário do protocolo HTTP onde um cliente faz um pedido ao servidor e o servidor apenas pode responder, com o protocol *WebSocket* o cliente e o servidor podem manter uma ligação activa e trocar pacotes entre si sem a necessidade de fazer pedidos por novos pacotes.

Como este protocolo corre sobre o protocolo TCP, garante a mesma confiabilidade, ordem correta de pacotes e deteção de erros nos pacotes, e como não necessita de plugins adicionais, é uma tecnologia perfeita para monitorizar a utilização do serviço.

Até à data de Outubro de 2016, o protocolo *WebSocket* era suportado por 90% dos *browsers* ativos [4], o que ajuda na criação de um serviço compatível com o maior número de dispositivos possível.

Esta tecnologia vai ser utilizada para manter uma ligação aberta entre o servidor e a página do administrador para a gestão do serviço, permitindo assim ver em tempo real a utilização das *gateways* por parte dos clientes.

2.2.2 WebRTC

WebRTC é uma API que torna possível fazer chamadas de voz e vídeo e partilhar ficheiros diretamente entre dois utilizadores sem a necessidade de *plugins*.

Como esta API corre de forma nativa nos *browsers*, não há necessidade de criar *software* especializado para criar aplicações de comunicação, e como suporta comunicação ponto-a-ponto (P2P), também não é necessário ter servidores especializados para trocar dados entre utilizadores.

O WebRTC é também um competidor dos operadores de telecomunicação tradicionais, uma vez que funciona na maioria dos *smartphones* e não necessita de *software* adicional. Além disso, tem como vantages: não tem custos

extra no caso das chamadas internacionais, não está dependente do fornecedor de *Internet* e não possui limitações associadas aos planos contratados aos fornecedores de telecomunicação.

Embora não seja necessário ter servidores especializados para que dois utilizadores possam comunicar, é necessário que haja uma forma de estes utilizadores se encontrem, e isto é feito utilizando um servidor de sinalização. A sinalização permite encontrar utilizadores e trocar informações de sessão entre eles (como o tipo de conteúdo multimédia que suportam, informação que pretendem transferir e protocolos de comunicação), para que possam estabelecer uma ligação entre eles para comunicarem.

Para estabelecer a comunicação entre dois utilizadores (**Figura 2.8**), estes necessitam de estar alcançáveis pelo servidor de sinalização, e para isso é necessário que acedam acedam a uma página *web* específica. Ambos os utilizadores acedem a essa página e um utilizador escolhe o outro na lista de contactos para pedir para iniciar a ligação. O *browser* do destinatário emite uma notificação e quando este aceita a chamada, é estabelecida uma ligação e é transmitido áudio e/ou vídeo. Para isto ser possível, quando o emissor inicia a ligação o seu *browser* gera a sua informação local na rede, esta é transmitida para o recetor que, quando aceita a chamada, guarda guarda-a como informação remota e gera a sua informação local, que é enviada para o emissor para ele a guardar como informação remota. A ligação é estabelecida e a informação negociada (áudio e/ou vídeo) é transmitida.

Até à data de Agosto de 2016, WebRTC era suportado por 62% dos *browsers* ativos [5].

ICE

o WebRTC ainda possui algumas limitações devido à forma como o NAT funciona, o que pode impossibilitar a comunicação P2P entre dois utilizadores, uma vez que estes podem não ter conhecimento do ser endereço público na *Internet*. ICE (*Interactive Connectivity Establishment*) é um protocole que facilita a comunicação entre máquinas dentro das suas redes privadas diferentes (que até podem estar atrás de *firewalls*). É utilizado para fazer percorrer as camadas de NAT que podem existir, utilizando para o efeito uma combinação de métodos que inclui STUN e TURN, a seguir explicados.

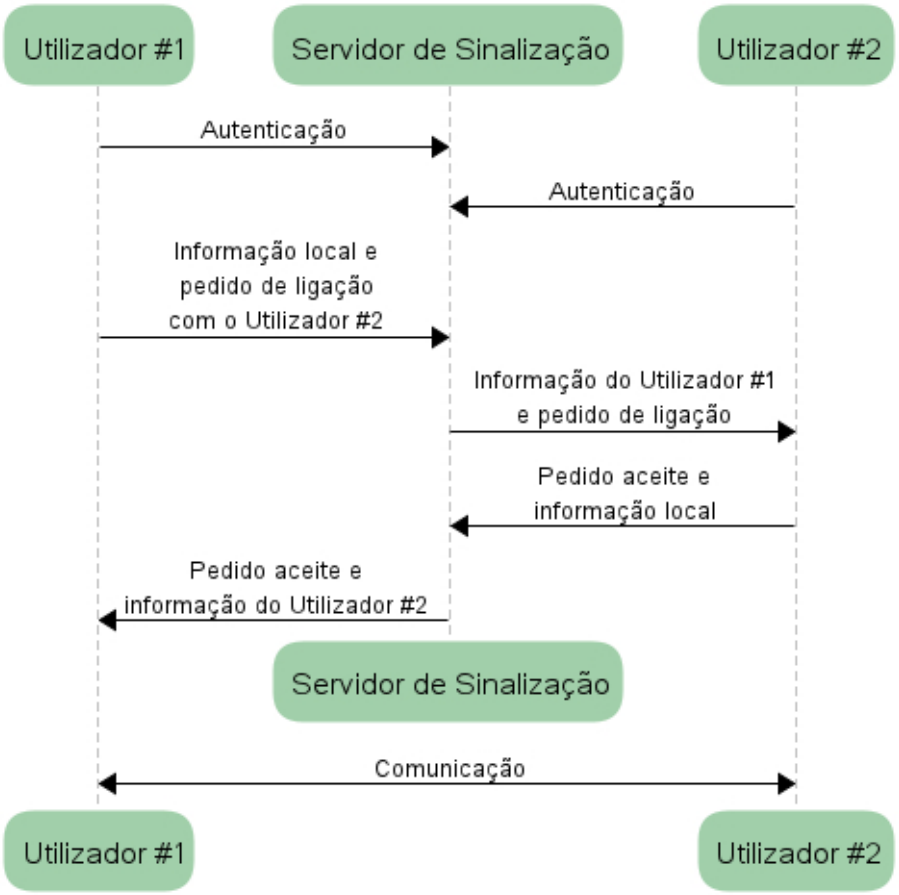


Figura 2.8: Estabelecimento da comunicação entre dois utilizadores

STUN

STUN (*Session Traversal Utilities for NAT*) é um protocolo dá a máquinas que estão a operar atrás de NAT a possibilidade de estabelecer um canal de comunicação com uma máquina que esteja fora da rede local.

Um servidor STUN permite aos utilizadores descobrirem:

- O seu endereço público
- O tipo de NAT em que se encontram
- O porto público que foi associado ao seu porto local pelo NAT

Para obter o utilizador obter a sua informação pública **(Figura 2.9)**, este faz um pedido ao servidor STUN através da sua *gateway* NAT, o servidor STUN encapsula o endereço público (para não ser traduzido pelo NAT) e envia-o para o utilizador.

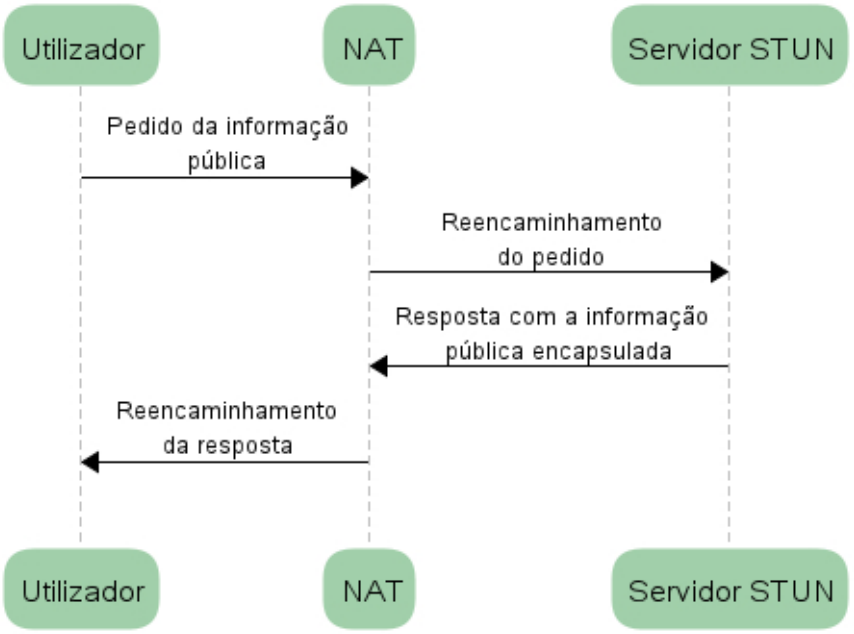


Figura 2.9: Comunicação entre um utilizador e o servidor STUN

Ao utilizar o servidor STUN (ou servidores, uma vez que os utilizadores não necessitam de utilizar o mesmo) **(Figura 2.10)**, os utilizadores têm na sua posse a informação necessária para trocar entre si (através do servidor de sinalização) e estabelecer um canal de comunicação direto.

TURN

O **TURN (*Traversal Using Relay NAT*)** foi desenvolvido para ultrapassar as limitações do STUN (o facto de não ser capaz de atravessar NATs simétricas).

O STUN possui uma limitação: o facto de não ser capaz de atravessar NATs simétricos. Para ultrapassar esta limitação foi desenvolvido o **TURN (*Traversal Using Relay NAT*)**, que funciona tendo um *proxy* para transmitir o conteúdo multimédia entre os utilizadores. O protocol TURN funciona sobre o STUN (para descobrir e trocar a informação local e remota dos utilizadores), com diferença de funcionar como intermediário de conteúdo multimédia entre os utilizadores. Isto significa que é possível ter um único servidor a atuar como servidor STUN e TURN.

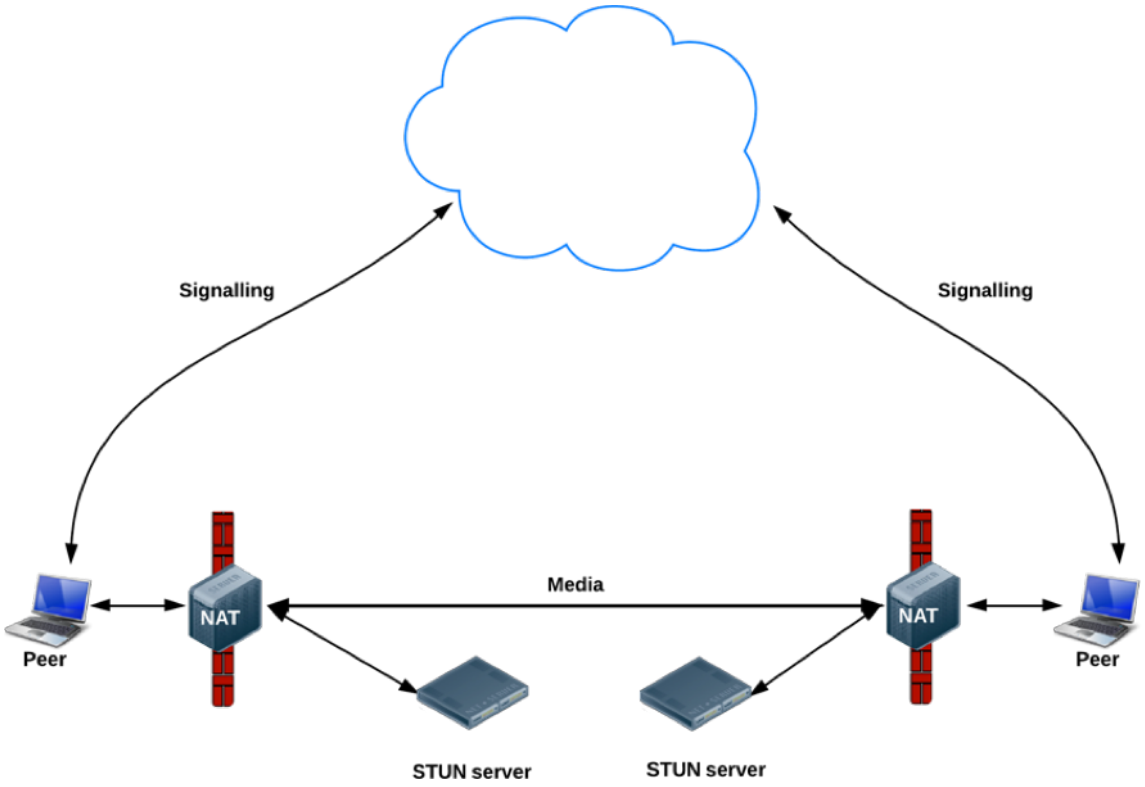


Figura 2.10: Comunicação entre dois utilizadores atrás de NAT utilizando servidores STUN [1]

Num cenário com servidores TURN **(Figura 2.11)**, os utilizadores (atrás de NAT) começam por comunicar com os servidores STUN para obter a informação pública, trocam essa informação através do servidor de sinalização e criam o canal de comunicação (que passa pelos servidores TURN).

STUN vs TURN

A grande diferença entre os protocolos STUN e TURN é que num cenário sem servidores TURN o conteúdo multimédia é transmitido diretamente entre os utilizadores, enquanto que num cenário com servidores TURN estes funcionam como uma ponte na transmissão de conteúdo entre os utilizadores. Isto dá ao TURN grandes desvantagens como o custo e a largura de banda, mas é um meio necessário quando os utilizadores estão sujeitos às restrições do NAT.

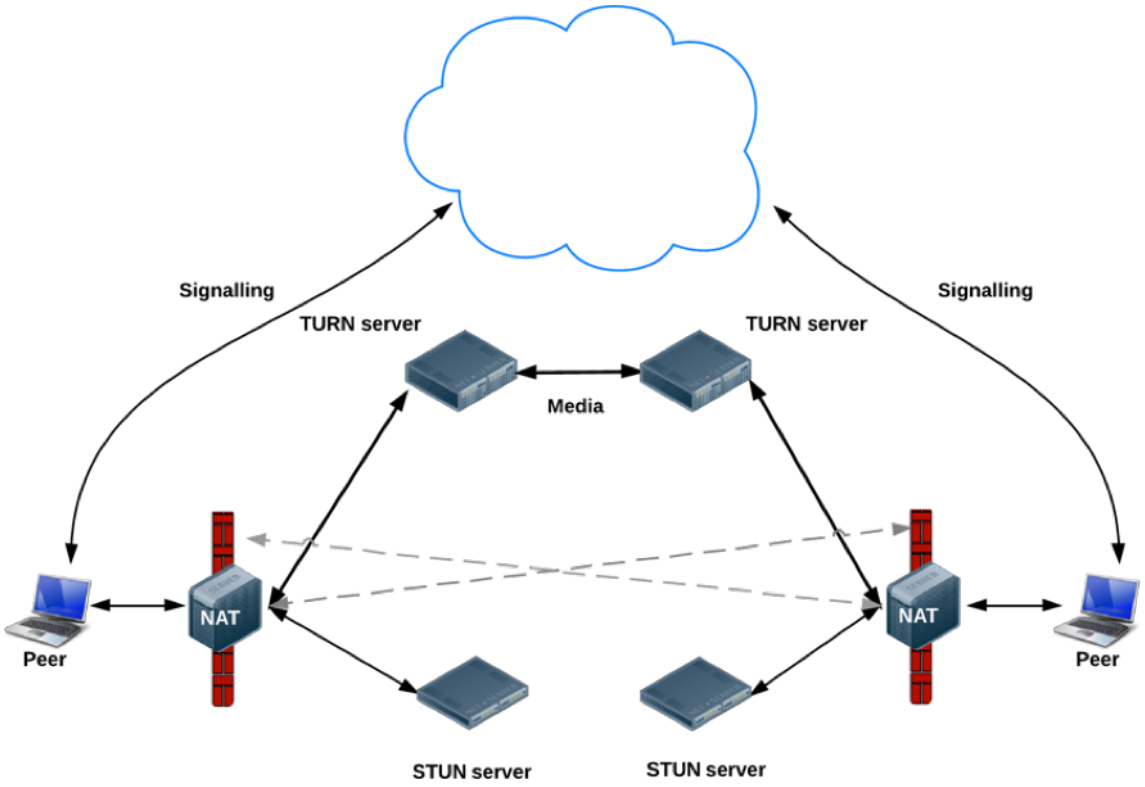


Figura 2.11: Comunicação entre dois utilizadores atrás de NAT utilizando servidores STUN e TURN [1]

2.2.3 Balanceamento de carga geográfico

O balanceamento de carga geográfico (*Geo* DNS) é um tipo de balanceamento de carga de DNS que permite balancear e encaminhar os pedidos dos utilizadores com base na sua localização geográfica. Assim, quando um utilizador faz um pedido de DNS (para encontrar o IP de um determinado serviço) a um servidor DNS que suporte *Geo* DNS este vai devolver o endereço da instância disponível que está geograficamente mais perto do utilizador que fez o pedido.

O maior benefício deste tipo de escalonamento de DNS é a menor latência do serviço, o que é muito importante num serviço onde é transmitido conteúdo multimédia.

No entanto, embora esta arquitetura proporcione um alto desempenho das instâncias de um serviço, o balanceador de carga é um ponto crítico do sistema e, no caso de falhar, todo o serviço passa a estar inacessível. Por isso, é preferível utilizar um serviço de DNS dedicado em vez de utilizar um servidor de DNS privado, uma vez que iria necessitar de ser replicado e estar altamente disponível, algo que os serviços de DNS dedicados já oferecem.

Como este projeto faz uso da WIT WebRTC Gateway, que se integra com infraestruturas de telecomunicação, ter diversas instâncias da *gateway* em diferentes servidores espalhados pelo mundo e utilizar um balanceador de carga geográfico permite reduzir a latência das comunicações, uma vez que os servidores SIP dos clientes passam a comunicar com a instância da *gateway* WebRTC que se encontra mais perto deles.

Existem algumas soluções confiáveis que oferecem o serviço necessário, com as mais importantes a serem apresentadas a seguir.

Amazon Route 53

O **Amazon Route 53** é um serviço de DNS altamente disponível e escalável com suporte para *Geo* DNS. Permite encaminhar utilizadores para diferentes instâncias de um serviço, e ao mesmo tempo monitoriza essas instâncias e age como uma balanceador de carga de forma a reduzir a carga das instâncias.

O serviço pode ser configurado para agir de acordo com diferentes configurações, tais como: saúde (estado de uma instância que pode ser influenciado pela capacidade da máquina, erros de sistema, carga e estado do *hardware*), latência, carga e localização geográfica das instâncias. A configuração é feita através de uma interface gráfica chamada **Amazon Web Services Management Console**.

Este serviço é pago e os preços são apresentados na **Tabela 2.2**. Os preços são fixos em dólares americanos, sendo que a conversão da moeda é feita na altura do pagamento.

Tipo de encaminhamento	**Preço por milhão de pedidos/mês**	
	até mil milhões	**acima de mil milhões**
Normal	0.40 $USD	0.20 $USD
Com base na latência	0.60 $USD	0.30 $USD
Com base na localização	0.70 $USD	0.35 $USD

Tabela 2.2: Preços do Amazon Route 53

Google Cloud DNS

O **Google Cloud** DNS é outro serviço de DNS altamente disponível com suporte para *Geo* DNS. Tem caracteríetica semelhantes ao da Amazon, mas a

gestão é feita numa linha de comandos utilizando comandos próprios para o serviço, ou então criando aplicações que façam uso da API REST disponibilizada.

O preço deste serviço está disponível na **Tabela 2.3**, e é indiferente do tipo de encaminhamento.

Tipo de encaminhamento	**Preço por milhão de pedidos/mês**	
	até mil milhões	**acima de mil milhões**
Todos	0.40 $USD	0.20 $USD

Tabela 2.3: Preços do Google Cloud DNS

BIND com a extensão GeoDNS

O **BIND** é um *software* de código aberto que implementa os protocolos DNS. Está dividido em três partes:

- ***Domain Name Authority***: servidor que devolve informação como endereços IP e outros registos
- ***Domain Name Resolver***: servidor que faz pedidos recursivamente a outros *Domain Name Resolvers* até encontrar o *Domain Name Authority* correto
- **Ferramentas**: conjunto de ferramentas operacionais e de diagnóstico

O GeoDNS é uma extensão para o BIND que adiciona a filtros com a capacidade de fazer o escalonamento com base na localização de quem fez o pedido. A deteção da localização é feita com recurso a bases de dados com mapeamentos de países e endereços IP, nomeadamente o MaxMind GeoLite (solução grátis) e o GeoLite GeoIP (solução paga, com mais detalhes ao nível de cidades e estados).

Escolha do balanceador de carga geográfico

Se fosse escolhido o BIND, era necessário instalar e configurar o BIND numa máquina e posteriormente replicá-la para não se tornar um ponto crítico, algo que já está implementado nos serviços da Amazon e Google. Como isto ia levar a um aumento desnecessário da complexidade do projeto, esta solução

foi descartada, e foi escolhido o serviço Amazon Route 53 por oferecer um bom serviço e fácil de configurar.

2.2.4 Serviços de pagamentos pela internet

Um serviço pago necessita de disponibilizar meios de pagamento confiáveis para que os clientes possam efetuar pagamentos de forma segura, fácil e rápida. Existem diversos serviços de pagamentos pela internet compatíveis com um grande número de moedas e autorizados legalmente a operar num grande número de países. Nesta secção são analisados os serviços mais conhecidos, que são aqueles que não só têm um maior número de utilizadores mas também são os que transmitem uma maior confiança quando um novo utilizador necessita de criar uma conta.

PayPal

O **PayPal** é o serviço de pagamentos pela Internet mais utilizado. É dos serviços mais fáceis de utilizar, permitindo o pedido de pagamentos e transferência de fundos entre contas utilizando um endereço de *email*, e permite ainda associar cartões de crédito e fazer transferências diretamente a partir das contas bancárias dos clientes.

A 2 de Agosto de 2016, o PayPal operava em 202 países e era compatível com 25 moedas, tendo um total de 188 milhões de utilizadores [6].

Do ponto de vista de desenvolvimento de *software*, o PayPal oferece uma fácil integração em projetos através do SDK que disponibilizam, permitindo que os utilizadores façam pagamentos sem sair das aplicações ou encaminhado o os utilizadores para a página do Paypal, deixando que este trate da interação com o utilizador no ato de pagamento de um serviço.

Google Wallet

O *Google Wallet* é um serviço de pagamentos pela Internet que está em crescimento e que permite fazer transferência em contas Gmail, também pertencente ao seu leque de serviços, utilizando o endereço ou número de telefone associados. Tal como o PayPal, permite associar cartões de crédito e

contas bancárias, facilitando os pagamentos sem a necessidade de adicionar fundos ao Google Wallet.

A grande desvantagem em relação ao Paypal é a sua limitação geográfica, uma vez que, a 2 de Agosto de 2016, apenas permitia fazer transferências dentro dos Estados Unidos.

Serviço de pagamentos escolhido

Existem outros serviços de pagamentos com menor quota de mercado tal como o **Dwolla** (que permite fazer transferências através de email, telefone, Facebook, LinkedIn e Twitter, não aplica taxas em transações com valor até 10 dólares americanos), **Authorize.net** (que suporta cartões de crédito) e **We-Pay** (uma plataforma muito simples que permite associar cartões de crédito e contas bancárias, mas que apenas está disponível para clientes dos Estados Unidos), no entanto estes serviços são menos conhecidos pelos utilizadores, o que pode reduzir a confiança dos utilizadores num serviço que apenas permita fazer pagamentos através destas soluções.

O serviço escolhido para esta prova de conceito foi o PayPal por ser o serviço mais utilizado e conhecido. No entanto, como não é obrigatório oferecer apenas um meio para efetuar pagamentos, é possível vir a integrar outros serviços de pagamentos num futuro próximo, sendo que a escolha de novos serviços a integrar pode estar novamente dependente da sua popularidade e número de utilizadores.

2.2.5 Servidor *web*

A escolha de um bom servidor *web* tem um grande impacto no desenvolvimento de um projeto uma vez que certas tarefas como a disponibilização de ficheiros estáticos, configurações de segurança, mapeamentos de endereços, gestão de sessões e gestão de recursos já estão parcial ou completamente implementadas, requerendo apenas a integração com o projeto e o desenvolvimento do conjunto de funcionalidades que realmente interessam para o serviço.

Existem diversas soluções, sendo que se destacam o **NodeJS** (que permite correr uma aplicação servidor desenvolvida em JavaScript e que disponibiliza módulos que tratam do I/O, acesso a bases de dados, segurança via

SSL e concorrência) e o **Apache Tomcat** (capaz de correr aplicações Java EE, constituído por vários componentes responsáveis por escutarem ligações TCP, gerir recursos e gerir a distribuição de clientes).

A **Spring Framework** é um conjunto de ferramentas para a plataforma Java e que permite criar aplicações sobre o Java EE, e já integra o Apache Tomcat facilitando a criação de aplicações às quais os clientes se podem ligar. Disponibiliza vários módulos que permitem o rápido desenvolvimento de aplicações web, tais como:

- Autenticação: disponibiliza componentes de segurança com suporte para vários padrões e protocolos, com possibilidade para extender estes componentes.
- Acesso a conteúdo: disponibiliza uma fácil integração com bases de dados relacionais e não relacionais.
- *Model-view-controller* (MVC): disponibiliza componentes que permitem a configuração e extensão de aplicações web e serviços REST.
- Testes: possui suporte para a escrita de testes unitários e de integração.

A Spring Framework foi escolhida para o backend da aplicação devido às suas capacidades e grande suporte. O Apache Tomcat já integrado é uma solução estável para usar para interpretar os pedidos dos clientes e disponibilizar os ficheiros para construir a aplicação web do lado do cliente. Com base nos pedidos, o servidor faz a validação dos clientes e pedidos (com o auxílio dos módulos já existentes) e disponibiliza a informação de acordo os pedidos.

2.2.6 Base de dados

O projecto precisa de uma forma de armazenar a informação dos utilizadores para que esta possa ser facilmente acedida e gerida. Foi escolhida uma base de dados relacional (DBMS) uma vez que este tipo de base de dados permite manter a informação organizada por tabelas, permite manter relações entre elas através de chaves ou índices e permite forçar restrições quanto ao tipo de conteúdo a guardar (como o tipo e o tamanho dos campos). A maioria deste tipo de base de dados segue padrões SQL para aceder e armazenar informação, incluíndo tipos e métodos para tratar os dados.

Os DBMS mais conhecidos e confiáveis são: **PostgreSQL**, **MySQL** e **Oracle**. Estes apresentam funcionalidades e performance muito semelhantes, sendo que apenas se destacam em certos aspetos em cenários em que existe uma grande quantidade de dados e o número de acessos é elevado, que fazem com que as suas implementações de acesso concorrente, recuperação, optimização, replicação e escalabilidade sejam cruciais para que se possam destacam.

Em geral, as caraccterísticas deste tipo de base de dados são:

- Capacidade de replicação assíncrona
- Multi plataforma
- Compatíveis com múltiplos controladores (compatíveis com diferentes linguagens de programação)
- Compatíveis com os principios ACID (que garantem que apenas ações bem sucedidas são aplicadas e são persistentes, e que a base se encontra num estado válida após qualquer transacção)
- Disponibilizam acesso concorrente e controlo transacional
- Criam registos antecipados das modificações (para ser tolerantes a falhas)
- Altamente escaláveis, tanto em relação à quantidade de dados como ao número de acessos concorrentes
- Suporta funções, cursores e *triggers* (que permitem executar ações em função de outras ações)
- Recuperação da informação

Qualquer uma das soluções apresentadas seria uma boa escolha, mas como nesta prova de conceito não se vai chegar a um cenário que permita verificar diferenças críticas na performance dos diferentes motores de DBMS, a escolha não é crítica, e por isso foi escolhido o PostgreSQL por ter mais suporte dentro da empresa.

2.2.7 *Frameworks* para aplicações *frontend*

Uma *framework frontend* ajuda no desenvolvimento de aplicações *web* permitindo separar o projeto em três partes:

- Modelo de dados: comtém a informação a apresentar
- Visualização: constituída pelos ficheiros HTML e CSS que são utilizados para construir as páginas da aplicação e apresentar a informação (do modelo de dados) de uma forma organizada e lógica.
- Controladores: estabelecem a ligação entre o modelo de dados e os componentes da visualização, tratando da associação da informação e ações a executar.

A parte *frontend* do projeto vai ser constituída por vários ficheiros HTML com os modelos das páginas e dos componentes. A *framework* vai ser utilizada para comunicar com o servidor e utilizar os modelos e a informação para construir as páginas da aplicação.

De entre as várias *framework* existentes, aquelas que mais se destacam são apresentadas a seguir.

AngularJS

O **AngularJS** é uma *framework frontend* para aplicações *web* que apresenta uma arquitetura MVC, permitindo separar os componentes de lógica, dados e apresentação. É contruída sobre o JavaScript e permite criar *tags* HTML personalizadas para gerir o conteúdo das páginas. À semelhança do Ajax, permite enviar pedidos e receber informação do servidor sem a necessidade de voltar a abrir a página nem de a alterar, o que permite criar aplicações web mais interativas que apenas precisam de atualizar certas partes da página, e como cria uma abstração do DOM (a estrutura com os componentes da página *web*) os desenvolvedores penas têm de tratar de construir os modelos para apresentar a informação e *framework* trata de atualizar a página sempre que o modelo de dados é atualizado. Também suporta *dependency injection*, permitindo criar aplicações mais modulares sem a necessidade de criar código repetitivo para estabelecer as ligações entre os módulos.

O AngularJS usa uma sintaxe específica nos modelos HTML (como ciclos e condições), o que algumas pessoas considerem como desvantagens uma vez que requere a aprendizagem de uma nova syntaxe para poder ser utilizada.

As grandes vantagens do AngularJS são o mapeamento bidireccional, que permite associar o modelo de dados e os componentes da visualização e trata de atualizar uma das partes sempre que a parte associada é alterada, e o facto de possuir uma enorme comunidade a desenvolver aplicações com esta *framework*, que dá suporte, sugere alterações e disponibiliza módulos que ajudam na criação de novas aplicações.

React

O **React** é uma biblioteca JavaScript para a construção de interfaces de utilizador. Também cria uma abstração do DOM, disponibilizando um ambiente de desenvolvimento mais simples, e utiliza *tags* HTML personalizadas para apresentar a informação. O React usa a sintaxe do JavaScript e é integralmente interpretado pelo motor de JavaScript, descartando a necessidade de aprender uma nova sintaxe.

O React é mais rápido que o AngularJS, mas é mais focado na parte da visualização no modelo MVC e não permite mapeamento bidirecional entre o modelo de dados e a visualização, focando-se apenas em atualizar a visualização quando o modelo de dados é alterado e não permitir a atualização do modelo de dados a partir dos componentes da visualização.

AngularJS 2

O *AngulasJS 2* é a nova versão do AngularJS. Não compatível com a versão anterior, uma vez que a sintaxe e a estrutura foi alterada drasticamente, mas disponibiliza várias API que ajudam num eventual processo de atualização de uma aplicação. Continua a adicionar a sintaxe específica no código HTML, mas como é diferente da versão anterior há a necessidade de aprender essa sintaxe nova, mesmo para desenvolvedores familiarizados com a versão anterior.

O *AngularJS 2* apresenta um desempenho consideravelmente melhor que o seu antecessor, mas como ainda está numa fase beta pode não ser a escolha mais segura.

Escolha da *framework frontend*

Embora o AngularJS nao tenha a melhor performance quando comparado com as outras *frameworks*, continua a ser uma boa escolha onde para qualquer projeto onde a performance não seja crítica ou onde as diferenças de performance não sejam suficientemente perceptíveis para causar impacto. Este projeto não possui uma complexidade que torna perceptíveis as diferenças de performance, e por isso este aspecto não é o mais crítico na escolha da tecnologia.

Como o AngularJS 2 ainda está numa versão beta, não foi escolhido porque podem surgir problemas inesperados e com baixa prioridade na lista de problemas a corrigir. Assim, o AngularJS foi escolhido pelas suas capacidades (em especial o mapeamento bidirecional) e pelo suporte (dentro e fora da empresa).

2.2.8 WIT WebRTC Gateway

A WIT WebRTC Gateway é uma solução de comunicação da WIT Software que facilita o desenvolvimento de aplicações de comunicação usando WebRTC. Esta *gateway* faz uso das vantagens do WebRTC para permitir chamadas através de *browsers* sem *plugins* e fornece a conversão necessária entre o protocolo WebRTC e protocolos de telecomunicação.

Figura 2.12: Integração da WIT WebRTC Gateway na infraestrutura de comunicações [2]

A WIT WebRTC Gateway é um produto instalável na rede dos cliente, e integra-se com a sua infraestrutura de telecomunicações ligando-se à sua rede *IP Multimédia Subsystem*, passando a permitir chamadas entre *browsers* e aplicações que suportem WebRTC e dispositivos de telecomunicação. Para manter a compatibilidade com *browsers* que não suportem WebRTC, a gateway possui uma solução de reserva que utiliza o protocolo Flash RTMP para a transmissão de áudio e vídeo.

A gateway já integra *proxies* de sinalização e transmissão multimédia, e apresenta várias funcionalidades como:

- Livro de contactos
- Serviço de gestão de mensagens
- Monitorização
- Estatísticas
- Sessões HTTP
- Gestão do sistema
- Segurança

Capítulo 3

Metodologia

O desenvolvimento de *software* é uma área que está em constante mudança, e com tantos competidores o produto de um projecto pode ficar obsoleto mesmo antes de chegar a ser concluído. A WIT tem vindo a adoptar um processo de desenvolvimento iterativo e incremental baseaso em metodologias ágeis. O processo adotado não requer que o planeamento no início de um projecto seja rigoroso em relação os tempos a seguir, uma vez que é extremamente difícil seguir um plano a longo termo. Além disso tem em conta o facto de as pessoas não serem perfeitas a fazer estimativas do tempo que se demora a concluir as tarefas, e como qualquer projeto está sujeito a situações inesperadas e como há competidores a trabalhar em soluções semelhantes, um projeto tem que se adaptar de forma a conseguir aguentar contra os concorrentes.

3.1 Scrum

O **Scrum** é um grupo de métodos de desenvolvimento de *software*, para gerir o desenvolvimento de produtos, onde a equipa trabalha como um todo para atingir um determinado objetivo. Com Scrum é necessário fazer o levantamento da lista de funcionalidades para o projecto, priorizá-las e distribuí-las por *sprints*.

Um *sprint* é um bloco de tempo durante o qual é suposto adicionar novas funcionalidades e obter uma iteração estável do produto, com potencial para ser entregue e ser considerada uma versão final.

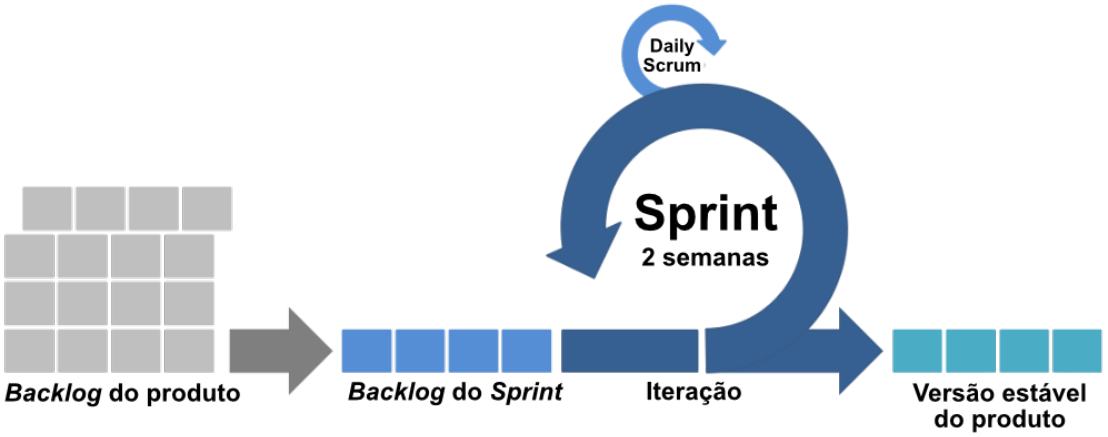


Figura 3.1: Processo do Scrum [3]

Além da parte do desenvolvimento do produto, um *sprint* engloba também outros quatro eventos:

- ***Sprint Planning***: é feito no início do *sprint*, e é onde a equipa planeia o âmbito desse *sprint*, escolhendo para o efeito as tarefas, do *backlog* do produto (a seguir explicado), a desenvolver durante esse *sprint*.
- ***Daily Scrum***: reunião rápida onde cada membro fala do que foi que foi feito no dia anterior para cumprir os objetivos do *sprint*, o que planeiam fazer durante esse dia e se há impedimentos que possam impedir algum membro de alcançar o objetivo.
- ***Sprint Review***: feita no fim do *sprint*, é onde a equipa revê o trabalho concluído e o trabalho planeado que não foi possível concluir, e onde é apresentado o trabalho concluído aos *stakeholders* (interessados no produto do projecto).
- ***Sprint Retrospective***: também feita no fim do *sprint*, é onde a equipa reflete sobre o *sprint* terminado e ajuda na melhoria do processo de desenvolvimento (em aspetos como o ritmo de trabalho e estimativas das tarefas).

O Scrum define três papéis principais na equipa:

- ***Product owner***: representa os *stakeholders* e é responsável por reunir, avaliar e priorizar os requisitos do ponto de vista dos clientes.
- ***Development team***: responsável por adicionar novas funcionalidade ao produto e entregar uma versão estável do produto no fim de cada *sprint*.

- ***Scrum master***: ajuda a equipa a respeitar os práticas e metodologias do Scrum, e é responsável por remover qualquer impedimento que impeça a equipa de atingir os objetivos do produto.

O facto de o projeto ser desenvolvido iterativamente e ser obtido um produto estável com novas funcionalidade no fim de cada *sprint* ajuda a equipa a adaptar-se mais facilmente quando os requisitos mudam. E quando é necessário apresentar o trabalho atual e o *sprint* atual não está concluído ou o projeto contém erros, é sempre possível obter a última versão estável e apresentar um produto a funcionar corretamente.

3.1.1 Processo de desenvolvimento

3.1.1.1 *Backlog*

Na metodologia Scrum não é necessário um documento tradicional com os requisitos do produto. Em vez disso é produzido um *backlog* do produto.

O *backlog* do produto é um artefacto que contém um conjunto de requisitos de alto-nível que devem ser atingidos no âmbito do projeto. É definido sob a forma de *user stories* (explicadas no **Capítulo 3.1.1.3**), que são posteriormente priorizadas pelo *product owner* e estimadas pelos membros da equipa.

O *backlog* do *sprint* é um conjunto de funcionalidades, escolhidas do backlog do produto no início de cada *sprint*, que a equipa de desemvolvimento planeia completar durante esse *sprint*. Essas funcionalidades são então divididas e especificadas sob a forma de tarefas, que vão sendo marcadas pelos membros da equipa como estando em desenvolvimento ou completas, durante a fase de desenvolvimento do *sprint*.

3.1.1.2 *Definition of Done*

A ***Definition of Done*** é uma lista de requisitos explicados de forma clara e concisa e que explicam o que uma funcionalidade e uma iteração do produto devem respeitar para poderem ser consideradas como completas. Esta lista é definida no início do projeto e coloca a equipa a concordar no que deve ser feito para concluir uma tarefa ou iteração do produto, ajudando a equipa a

decidir o esforço necessário para cada tarefa e a quantidade de trabalho que deve ser atribuída a um *sprint*.

3.1.1.3 *User stories*

Uma ***User story***, que define uma possível ação do utilizador quando utiliza o produto, é uma forma de representar uma funcionalidade. Esta identifica um requisito do produto numa forma perceptível por qualquer pessoa. A história de utilizador pode depois ser dividida em múltiplas tarefas que especificam o que o produto deve ser capaz de fazer para permitir ao utilizador executar essa ação.

Uma historia de utilizador é definida numa das formas:

- "Como *[tipo de utilizador]* quero *[acção pretendida]*."
- "Como *[tipo de utilizador]* quero *[acção pretendida]* para *[motivo da ação ou resultado a atingir]*."

Uma história de utilizador é terminada quando respeita a *Definition of Done* das *user stories*.

3.1.1.4 Reunião de *sprint*

A reunião de *sprint* ocorre a cada duas semanas e é onde os membros da equipa apresentam o trabalho desenvolvido durante esse *sprint* ao *product owner*. Isto é feito numa pequena apresentação onde é relembrado o que era suposto fazer durante esse *sprint*, o que foi concluído, o que não foi concluído e porquê, seguido por um demo da aplicação com foco nas novas funcionalidades. O trabalho é finalmente revisto e são escolhidas novas funcionalidades para o *sprint* seguinte.

Neste projeto a duração dos *sprints* era de duas semanas. No fim de cada *sprint* havia uma reunião onde o auto apresentava o progresso do projeto após esse *sprint*, o que foi concluído e os problemas encontrados, e apresentava uma demo do projeto com foco nas novas funcionalidades. A reunião terminava com a avaliação do trabalho realizado e com a preparação do *backlog* para o novo *sprint*.

3.1.2 Aprendizagem

A abordagem do scrum ajuda a equipa manter o projeto bem orientado e minimizar erros. Qualquer projeto está sujeito a erros de estimativas (de tempo ou carga), e a sucessiva acumulação de atrasos nas tarefas pode impossibilitar a conclusão de um projeto a tempo. Com o Scrum, a equipa aprende o seu ritmo de trabalho e as estimativas feitas inicialmente vão sendo atualizadas para refletirem esse ritmo, resultando em estimatívas cada vez mais precisas para o projeto atual e para os seguintes. Isto ajuda a equipa a tomar melhores decisões num projeto, como a necessidade de adicionar novos membros à equipa, filtrar as funcionalidades que devem constar no projeto final, ou dividir tarefas complexas em pequenas tarefas mais específicas.

3.2 Requisitos

Como neste projeto o *product owner* é o orientador do estágio pela empresa, e o autor é o único membro da equipa, este fica com os papéis de *Scrum master* e equipa de desenvolvimento, trabalhando diretamente no processo de desenvolvimento e reunindo com o *product owner*. O *product owner* fica responsável por avaliar o progresso do projeto e planear os sprints.

Antes do processo de desenvolvimento foi definido o *backlog* do produto. Para criar o *backlog* do produto o *product owner* apresentou os requisitos do projeto e o autor definiu a lista de *user stories*.

Os requisitos de alto nível inicialmente propostos foram:

- Configuração da *gateway* e SDK
  - Suporte de contadores numa base de dados centralizada
  - Suporte de chaves da API numa base de dados
- Promoção do serviço e página de registo
  - Página com informação do serviço (incluindo o design na interface)
  - Geração do SDK personalizado e configurado
  - Módulo para registo de contas e geração de chaves da API
  - Formulário de registo

- *Backoffice* para a gestão do serviço
  - Servidor para gerir controlar *backoffice*
  - Interface para o *backoffice* para gerir as configurações e pagamentos
  - Integração do serviço de pagamentos
- Implantação do sistema
  - Configuração do balanceador de carga geográfico
  - Instalação do *backoffice* e gerador do SDK

A lista de *user stories* produzida foi depois avaliada, aprovada e priorizada pelo *product owner*. O *backlog* to produto foi registado numa plataforma web para gestao de projetos (Redmine), onde podia ser seguido. A lista produzida está disponível no **Anexo B**.

Com o *backlog* produzido ficaram definidas as tarefas a desenvolver no projeto.

3.3 Planeamento

O planeamento foi feito para dois semestres, com o primeiro mais focado no estudo de soluções semelhantes e no estudo de tecnologias a utilizar, o segundo foi mais focado no desenvolvimento e execução de testes.

3.3.1 Primeiro semestre

No primeiro semestre foram estudados os requisitos de alto nível para obter a funcionalidades a implementar no protótipo. Com base nos requisitos foi feito um estudo de soluções semelhantes, com foco nas suas funcionalidades, e foi feita uma comparação para estudar os requisitos mais importantes.

Com base na informação das funcionalidades foi feito um estudo das tecnologias que podiam ser utilizadas no desenvolvimento do *backend* e do *frontend*, e foi feita a escolha das tecnologias a utilizar, e foi feito um estudo dos serviços a integrar com o *backoffice* (serviço de pagamentos e balanceamento de carga geográfico). Ainda com base nas funcionalidades foram criados *mockups* com a estrutura das páginas da plataforma *web* e as suas funcionalidades.

Por fim foi criada uma base para o *backoffice* de gestão do serviço, com a configuração necessária para incluir *endpoints* REST com os quais a plataforma *web* iria comunicar, e a partir desta configuração foram implementadas as funcionalidades de registo de contas e geração de chaves do serviço.

3.3.2 Segundo semestre

O segundo semestre começou com a especificação da UI da plataforma. Esta foi especificada com a ajuda de uma *designer*, com base nas *user stories* e nos *mockups*, e foi posteriormente aprovada pelo *product owner*.

As especificações da UI foram aplicadas às funcionalidades anteriormente implementadas, e seguiu-se a implementação das restantes funcionalidades da plataforma, relacionadas com a gestão de contas e a gestão de chaves.

Para a geração do SDK foi feita a alteração do *script* original de geração do SDK para permitir a escolha dos módulos a incluir, e para adicionar a aplicação demo com a configuração dos cliente. Com o *script* funcional, foi feita a integração com o *backoffice* para este poder iniciar a geração do SDK quando pedido por um cliente.

A seguir foi integrado o serviço do PayPal com base no SDK fornecido, e foi configurado o serviço Amazon Route 53 na plataforma disponibilizada.

A fase de desenvolvimento terminou com a alteração ao código das gateways para as integrar com o *backoffice* de gestão criado.

A fase de testes foi uma execução de todos os testes funcionais que foram sendo criados e testados ao longo do desenvolvimento, sendo estes descritos no **Capítulo 5**.

3.4 Riscos

Qualquer projeto de *software* está sujeito a riscos com diferentes origens. É necessário estar preparado e ter estratégias de prevenção ou mitigação de forma a que o impacto dos riscos no projeto seja minimizado, e estas estratégias podem ter que vir a ser revistas e alteradas durante o desenvolvimento uma vez que as tecnologias estão em constante mudança e o projeto pode encontrar novos riscos.

Os riscos detetados foram:

- **Utilização de novas tecnologias**
  - A utilização de novas tecnologias e frameworks pode ter impacto no tempo de desenvolvimento e resolução de problemas.
  - **Plano de mitigação**: na fase de investigação estudar as tecnologias e frameworks e desenvolver aplicações de teste para conhecer melhor as funcionalidades.
- **Más estimativas**
  - Algumas funcionalidades podem necessitar de mais tempo que o inicialmente esperado, e a necessidade de prolongar o tempo das tarefas devido a más estimativas pode fazer com que se torne impossível de terminar todas a funcionalidades pretendidas.
  - **Plano de mitigação**: manter o *backlog* do produto atualizado, atualizar estimativas e repriorizar funcionalidades, se necessário.
- **Competidores no mercado**
  - Os competidores atuais, e novos que possam surgir, aumentam a dificuldade de um produto se impôr no mercado e de se tornar um serviço conhecido e utilizado.
  - **Plano de mitigação**: compreender e satisfazer os requisitos do mercado, criar um produto sólido e inovar. Também é necessário estar atento a novos requisitos do mercado e aos produtos dos competidores.
- **Súbito crescimento do número de utilizadores**
  - O serviço pode ter um crescimento inesperado do número de utilizadores, o que pode levar a que fique indisponível.
  - **Plano de mitigação**: monitorizar o sistema e tornar a criação de instâncias do sistema uma tarefa fácil e rápida, permitindo aumentar os recursos disponíveis sempre que necessário. Opcionalmente, tornar o sistema compatível com serviços de computação na *cloud* para um rápido aumento de recursos.

Capítulo 4

Arquitetura

A arquitetura de um sistema é um componente essencial de qualquer projeto de *software* uma vez que define a estrutura e comportamento dos vários componentes que constituem o sistema (tanto o comportamento interno de cada componente como a forma com os componentes comunicam entre si), servindo como guia para a escolha das tecnologias a utilizar, para a implementação do sistema, e posteriormente facilita o desenvolvimento no futuro, quer seja feito pela equipa original ou por outra.

Neste capítulo é apresentada a arquitetura do sistema responsável pela gestão do sistema e disponibilização do serviço de comunicação (*backend*) e a arquitetura da plataforma que fornece aos clientes e administradores o acesso e a configuração do serviço (*frontend*).

4.1 Arquitetura do sistema

Nesta secção é apresentada a descrição da arquitetura do sistema responsável por gerir o serviço (subscrição e comunicação) e por disponibilizar a plataforma que na qual o cliente pode fazer a subscrição do serviço. Esta secção está dividida em três grupos:

- **Visão geral**: apresenta uma visão geral do sistema, que inclui a comunicação entre as camadas do sistema (camadas de gestão e comunicação) e como os clientes interagem com as mesmas.

- **Camada de gestão**: também identificada como ***backoffice*** ou **WaaS**, descreve de forma mais aprofundada a constituição da camada que permite gerir o serviço e a forma como os seus componentes interagem.
- **Camada de comunicação**: descreve a forma como é feita a comunicação dos clientes com as *gateways* e como estas fazem a gestão do serviço com recurso ao *backoffice*.

4.1.1 Visão geral

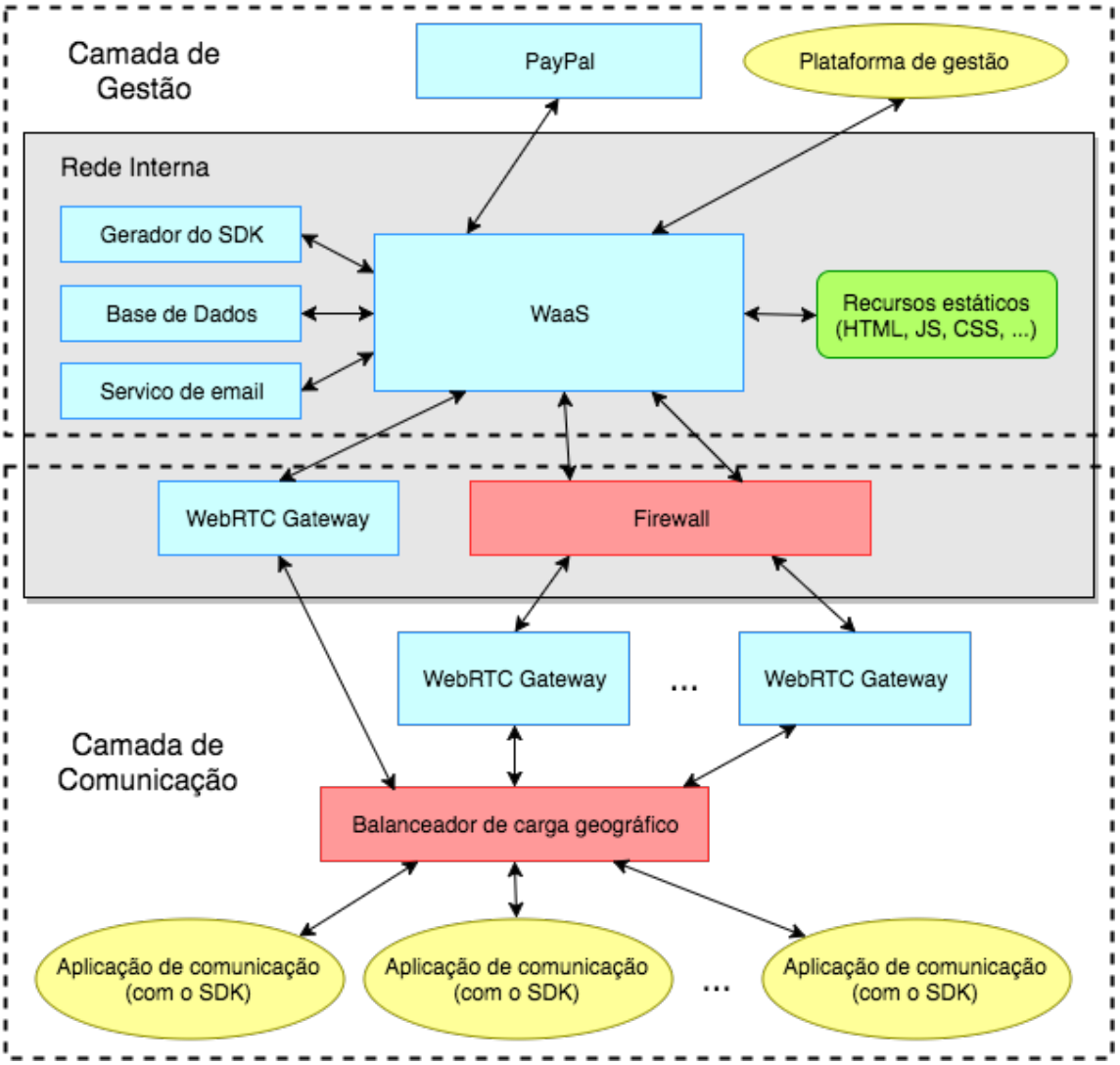
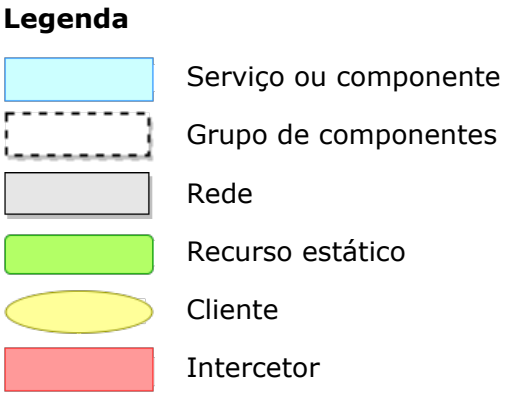


Figura 4.1: Arquitetura do sistema



O sistema é um conjunto de serviços e máquinas que permite a gestão do serviço e comunicação. Este apresenta uma camada de gestão que recebe os pedidos dos clientes (através da plataforma de gestão) e das *gateways*, trata da persistência da informação, através de uma base de dados que é descrita no **Capítulo 4.1.2**, e comunica com os serviços externos, nomeadamente o serviço de pagamentos (PayPal), o serviço de *email* e o gerador do SDK. Assim esta camada disponibiliza acesso a uma plataforma na qual os clientes podem subscrever o serviço de comunicação, e ao mesmo tempo fornece um ponto de acesso com o qual as *gateways* comunicam para fazer a gestão da utilização das chaves. A constituição desta camada é apresentada no **Capítulo 4.1.3**.

A camada de comunicação é constituída por uma ou várias *gateways* e um balanceador de carga geográfico. Os clientes comunicam com as *gateways* utilizando o SDK que obtiveram na plataforma de disponibilização do serviço. Nos caso das *gateways* que não estão na rede interna, os seus pedidos são validados pela *firewall*, onde devem ser inseridas regras para que estas tenham autorização para aceder ao módulo de gestão para fazer a validação das chaves. Esta camada é apresentada no **Capítulo 4.1.4**.

As *gateways* foram alteradas para validarem as chaves de serviço (API *Keys*) com a camada de gestão quando os clientes estabelecem a ligação a uma *gateway*. A gestão de utilização com base nas restrições das chaves também foi alterada para que essa informação fosse sincronizada com a camada de gestão para garantir que os clientes não abusam do serviço quando as suas comunicações estão distribuídas por diferentes *gateways*. Este processo é mais aprofundado no **Capítulo 4.1.4.1**.

4.1.2 Base de dados

A base de dados que armazena a informação do serviço relativamente às contas dos cliente é em PostgreSQL e é constituída por cinco tabelas, a seguir definidas:

- **Países**: guarda a lista de países com o nome do país em inglês e o identificador do país. Esta lista faz uso da biblioteca Country List, disponível em https://github.com/umpirsky/country-list com licença MIT.
- **Contas**: guarda a informação dos clientes e do administrador, incluindo informação de perfil e autenticação. Na informação do país apenas é guardada uma referência para a tabela de países.
- **Chaves**: guarda a informação das chaves, incluindo a chave secreta para utilização do serviço, estado, validade, capacidade, upgrades pendentes, e referência para o dono.
- **Confirmações**: guarda a informação das confirmações de ações, incluindo códigos de confirmação e cancelamento, data de criação e código identificador da ação.
- **Pagamentos**: guarda a informação de pagamentos de chaves, incluindo uma referência para a chave, identificadores do pagamento e do serviço de pagamento, estado, validade e capacidade da chave. A validade e capacidade contém os valores a adicionar à chave quando um pagamento é confirmado.

O diagrama físico da base de dados encontra-se disponível no **Anexo F**.

4.1.3 Camada de gestão

A camada de gestão é constituída por vários componentes responsáveis por diferentes tarefas de forma a reduzir a complexidade do sistema. Os componentes comunicam servem de ponte entre o controlador principal (módulo de gestão) e os componentes ou serviços externos.

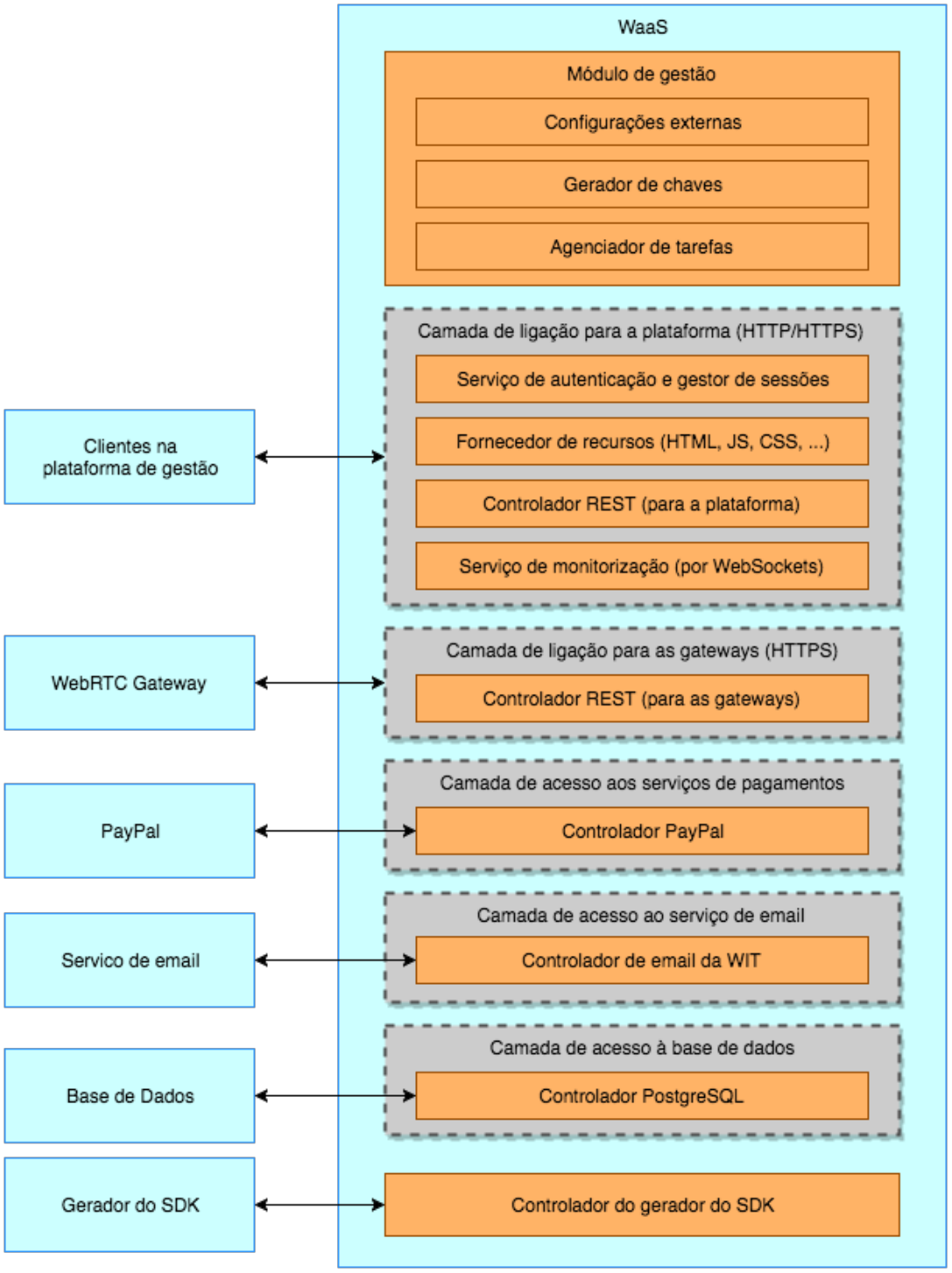
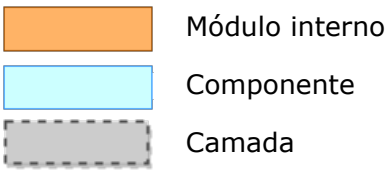


Figura 4.2: Arquitetura da camada de gestão

Legenda



Alguns componentes foram encapsulados em camadas que permitem fazer a abstração desses componentes para que possam ser substituídos por outras implementações, ou para que possam ser adicionadas novos componentes (como é o caso da camada de acesso aos serviços de pagamentos, que permite que sejam adicionados controladores para comunicar com outros serviços de pagamentos). Estes componentes são escolhidos e configurados sob a forma de *Java Beans* num ficheiro de configuração em formato XML.

Os componentes da camada de gestão são os seguintes:

- Módulo de gestão
- Configurações externas
- Gerador de chaves
- Agenciador de tarefas
- Camada de ligação para a plataforma
- Camada de ligação para as *gateways*
- Camada de acesso ao serviço de pagamentos
- Camada de acesso ao serviço de *email*
- Camada de acesso à base de dados
- Controlador do gerador do SDK

Módulo de gestão

Configurações externas

Este componente pertence ao módulo de gestão e disponibiliza os parâmetros configuráveis do sistema. Esta configuração está disponível num ficheiro de propriedades (mapeamento de parâmetros e valores) que é lido no arranque do sistema.

A lista de parâmetros configuráveis está disponível no **Anexo D**.

Gerador de chaves

O gerador de chaves permite gerar o (*key secret*) para uma chave, valor esse que é utilizado na autenticação das aplicações que usam o SDK com as *gateways*. Este componente é acedido quando uma chave é ativada (seja por ter sido feito um pagamento ou por ter sido manualmente ativada pelo administrador). A implementação deste gerador usa o gerador aleatório do UUID (Universally Unique Identifier), que é um tipo de identificador standard utilizado no desenvolvimento de *software*. Este identificador é uma chave de 128-bits e é geralmente representado em hexadecimal na forma:

```
XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

O *key secret* gerado é a representação em hexadecimal do UUID gerado. Este identificador não garante que não haja colisões, mas a probabilidade de isso acontecer é extremamente pequena. No entanto, é feita uma verificação de colisão, e no caso de ela existir é gerado um novo *key secret*, até ser encontrado um que seja único.

Esta implementação cumpre os requisitos principais de uma API Key:

- **Identificação**: identifica de forma única uma chave autorizada a aceder ao serviço (o que permite obter as configurações do serviço subscrito)
- **Autenticação**: tem o tamanho suficientemente grande para não poder ser facilmente adivinhada, o que permite e autenticar o utilizador (ou seja, garantir que quem a utilizou é o cliente legítimo)

Este gerador pode ser substituído por outras implementações, cujo interesse será o de aumentar o número de caracteres do *key secret* ou até mesmo alterar os caracteres permitidos (uma vez que na implementação atual apenas são usados caracteres hexadecimais).

Agenciador de tarefas

O agenciador de tarefas é responsável por agendar e executar certas tarefas periódicas. Este módulo permite que sejam adicionadas novas tarefas, bastando que sejam encapsuladas para respeitar uma *interface* que foi criada

e que permite identificar quando uma tarefa deve ser executada e iniciar a sua execução.

No estado atual apenas existe uma tarefa agendável: a tarefa responsável por desativar chaves que expiram. Quando o sistema inicia e após a execução da tarefa, o agendador verifica quando a próxima chave vai expirar a agenda a execução da tarefa para essa data e hora. Quando um utilizador cria uma nova chave, o módulo de gestão comunica com o agenciador de tarefas para verificar se a data de expiração dessa nova chave é anterior à data de agendamento da tarefa de expiração de chaves e reagendar a tarefa caso seja necessário.

Camada de ligação para a plataforma

A camada de ligação para a plataforma é o componente que disponibiliza um ponto de acesso onde os clientes se podem ligar, para fazer a gestão do serviço, através da plataforma. Esta camada está dividida em quatro grupos:

- Serviço de autenticação e gestão de sessões
- Fornecedor de recursos
- Controlador REST
- Serviço de monitorização

O **serviço de autenticação** serve para restringir o acesso à plataforma, garantindo que todos os utilizadores tenham acesso à página de divulgação do serviço, mas que apenas os utilizadores registados tenham acesso à parte da gestão do serviço, garantindo também que nenhum cliente tenha acesso às contas de outros utilizadores, quer para obter informação quer para gerir o serviço. No entanto, qualquer conta de administrador tem acesso ao serviço de todos os clientes, podendo aceder à sua informação e efetuar ações na gestão do seu serviço.

A autenticação está dividida em duas partes: a autenticação com o endereço de *email* e *password* e a validação de pedidos através de um ID de sessão.

A *framework* Spring agiliza o desenvolvimento do controlo do mecanismo de autenticação, fornecendo um sistema de gestão de sessões que trata do mapeamento dos ID de sessão aos clientes autenticados.

Na parte de autenticação por *email* e *password*, quando um utilizador faz o pedido de autenticação o serviço de autenticação comunica com o fornecedor de dados de autenticação e pede os dados de autenticação associados ao endereço de *email* que o utilizador submeteu. Se existir uma conta associada a esse *email* o fornecedor de dados de autenticação vai devolver o hash da *password* com que a conta foi registada, o *salt* gerado no registo e o estado da conta. Com base nesta informação o serviço de autenticação vai verificar se a conta está ativa e se a *password* submetida pelo cliente é válida e, em caso afirmativo, o sistema de gestão de sessões da Spring para mapear o ID de sessão do utilizador à conta associada aos dados de autenticação introduzidos.

Quando um utilizador autenticado faz um pedido ao servidor, o serviço de **gestão de sessões** identifica o cliente através do ID de sessão e assim o controlador que processar o pedido tem conhecimento do cliente que fez o pedido. Este serviço foi configurado com as permissões para os vários grupos de recursos (acesso aos recursos estáticos, acesso à API REST com permissões de administrador, cliente ou utilizador não autenticado), para que o controlador REST e o fornecedor de recursos não tenham a necessidade de verificar as permissões dos utilizadores nos pedidos.

O **fornecedor de recursos** está encarregue de disponibilizar os recursos que constituem as páginas da plataforma, bem como disponibilizar aos clientes os pacotes com o SDK que foram gerados.

O **controlador REST** é responsável por fazer o mapeamento dos endereços REST às ações a executar. A *Framework Spring* facilita a criação de controladores REST permitindo fazer este mapeamento diretamente nos métodos sob a forma de anotações, além de permitir fazer o mapeamento de variáveis nos endereços REST e parâmetros nos pedidos. Os endereços REST neste controlador são utilizados para a plataforma de gestão poder comunicar com o sistema e permitir assim fazer a gestão do serviço.

Para organizar este controlador, foi criada uma API REST que está documentada no **Anexo C**, com indicação dos endereços, descrição e parâmetros.

O **serviço de monitorização** permite monitorizar o uso das chaves de serviço nas várias *gateways*. Para isso, este serviço disponibiliza *endpoints* que os clientes podem subscrever, através da plataforma de gestão, que fazem uso da tecnologia *WebSocket* para apresentar em tempo real as atualizações relativas à utilização das chaves nas várias *gateways*. Os *endpoints*

disponibilizados permitem obter os dados da utilização atual ou receber apenas atualizações de estado, e esta informação tem indicação das chaves dos clientes, a o número total de chamadas em curso (para cada cada chave) e o número chamadas em curso em cada *gateway* para cada chave. As atualizações são enviadas sempre que uma *gateway* faz um pedido ao módulo de gestão para reservar ou libertar um canal de comunicação para uma chave, sendo que as atualizações incluem um *timestamp* e podem incluir dados de uma chave ou de várias (existe um limite mínimo de tempo entre envios para não sobrecarregar a plataforma).

Camada de ligação para as *gateways*

A camada de ligação para as *gateways* disponibiliza um ponto de acesso para as *gateways* para que possam comunicar com o módulo de gestão para fazer a validação das chaves. São disponibilizados *endpoints* REST que permitem obter a informação acerca das chaves (incluindo validade e capacidade), reservar um canal para uma chave e libertar um canal. Este módulo foi criado com base na documentação da *gateway*, e por isso a informação acerca da chave que é enviada contém os parâmetros necessários pelas *gateways*.

A API REST para as *gateways* está documentada no **Anexo C**, com indicação dos endereços, descrição e parâmetros.

Esta camada não apresenta nenhum mecanismo de autenticação ou validação para detetar se os pedidos provêm realmente de *gateways*. Como indicado no **Capítulo 4.1.1**, esta proteção deve ser implementada ao nível da *firewall*. O ponto de acesso está disponível num porto diferente do da plataforma, e pode ser alterado na configuração. Assim, este porto deve ser utilizado juntamente com os endereços IP das *gateways* para construir regras para a *firewall* que garantam que o acesso apenas é permitido às *gateways*.

Camada de acesso ao serviço de pagamentos

A camada de acesso ao serviço de pagamentos é um gestor que contém uma lista de componentes que comunicam com serviços externos de pagamentos, que podem ser escolhidos através de um identificador único do serviço.

Como inicialmente planeado, apenas foi implementada a integração com o serviço de pagamentos do PayPal, mas o gestor de pagamentos está preparado para uma possível integração de outros serviços. Para lidar com diferentes serviços, foi criada uma camada de abstração que os novos módulos (que comunicam com os serviços externos de pagamentos) devem respeitar.

O gestor de pagamentos e os serviços que integram os serviços de pagamentos são definidos no ficheiro XML de configuração de *Beans*. A comunicação entre o módulo de pagamentos e os serviços de integração é feita através dos seguintes métodos, que devem ser implementados pelos serviços de integração:

- **init**: inicia a configuração do serviço para ficar pronto a utilizar.
- **getPaymentInformation**: comunica com o serviço de pagamentos para gerar a informação necessária para efetuar um pagamento de uma chave. Esta informação inclui um endereço para onde o cliente é redirecionado para efetuar o pagamento.
- **getPaymentState**: verifica o estado de um pagamento.
- **getServiceName**: devolve o nome do serviço sob a forma de um item da lista de serviços autorizados.

Os serviços de integração incluem ainda um parâmetro com a localização de um possível ficheiro com configuração adicional para comunicar com os serviços de pagamentos, cujo valor é atribuído através do ficheiro de configuração de *Beans*.

Para integrar o o serviço do PayPal foi utilizado o SDK que o próprio PayPal disponibiliza, e foi criado um serviço de integração que respeita as regras definidas anteriormente. Foi adicionado um ficheiro com a configuração adicional para a comunicação com o serviço (e indicado no ficheiro de configuração dos *Beans*) e é feita a autenticação no método de inicialização. Com base na documentação do PayPal foram implementados os métodos para gerar a informação de pagamento (incluindo endereços para o redireccionamento entre o serviço do PayPal e a plataforma) e verificar o estado de um pagamento.

Camada de acesso ao serviço de *email*

O gestor de *email* comunica com o servidor externo de *email* para enviar as mensagens de *email*. Este controlador é configurado no ficheiro de configuração de *Beans*, para permitir a autenticação do controlador com o servidor de *email*. Para o envio de *email* o controlador começa por recorrer à configuração externa para obter o texto a apresentar no assunto e o *template* HTML a apresentar no conteúdo. Com o *template* e a informação de envio, o controlador substitui os *tokens* existentes (listados na E.1, e identificados por estarem entre duplas chavetas curvas) no *template* pelos valores corretos e comunica com o servidor de *email* para proceder ao envio.

Os tipos de *email* suportados são a confirmação de criação de conta e a partilha de chaves secretas, e os *tokens* suportados são apresentados no **Anexo E**.

O gestor de *email* é um módulo que pode ser configurado para comunicar com diferentes servidores de *email*, seja por pertencerem a diferentes fornecedores ou por utilizarem diferentes configurações para a autenticação. A solução apresentada permite comunicar com o serviço de *email* da Google (Gmail) e com o serviço de *email* da WIT Software, sendo suficiente substituir na configuração a lista de propriedades, mas na versão final apenas está presente o ficheiro de configuração para o serviço da WIT.

Camada de acesso à base de dados

O gestor de dados também apresenta uma camada de abstração constituída por um fornecedor do serviço armazenamento (disponibiliza acesso de escrita, leitura, atualização e eliminação de entradas) e vários fornecedores de informação que são responsáveis por persistir a informação no serviço de armazenamento e ler essa informação para a disponibilizar ao módulo de gestão.

Os fornecedores de informação incluem:

- **Contas**: faz a gestão do perfil das contas.
- **Chaves**: faz a gestão dos dados das API *Keys*.
- **Confirmações**: faz a gestão dos códigos para confirmação de ações.

- **Países**: permite aceder à lista de países.
- **Pagamentos**: faz a gestão da informação relativa ao pagamento de chaves.
- **Autenticação**: fornece os dados de autenticação de contas com base no *email* ou no identificador único do cliente.

Na implementação desta camada no projeto, foi feita a comunicação com uma base de dados em PostgreSQL, e o componente que comunica com a base de dados está configurado sob a forma de um *Java Bean* no XML de configuração de *Beans*. Os vários fornecedores de informação estão preparados para ler e escrever na base de dados (utilizando para o efeito o componente que comunica com a base de dados em PostgreSQL) e fazer a conversão dos modelos de dados em memória e os dados nas tabelas na base de dados.

A estrutura da base de dados está descrita no **Anexo F**.

Controlador do gerador do SDK

O gerador do SDK é um *script* que permite gerar um pacote (ficheiro .zip) com o SDK e com um cliente demo pronto a comunicar com as *gateways*, que posteriormente pode ser descarregado pelos clientes. O controlador do gerador do SDK está encarregue de executar o *script* que faz a geração do pacote, passando-lhe os parâmetros que permitem a sua configuração.

Para criar o pacote este componente faz a validação da configuração a utilizar na geração do SDK (validação de parâmetros permitidos, validação dos valores e proteção contra valores que pudessem ser interpretados pelo sistema operativo como comandos a executar), uma vez que os parâmetros `features` e `credentials` (a seguir descritos) contêm configuração que vem no pedido de geração do SDK por parte do cliente e o pedido podia ser manipulado para tentar executar comandos de sistema.

Este componente mantêm um registo dos SDK gerados e respetiva configuração, para não voltar a executar o *script* de geração quando um cliente faz um pedido com a mesma configuração.

O *script* de geração do SDK é executado com o *grunt* e inclui quatro parâmetros:

- **gatewayAddress**: endereço comum às *gateways*. O *script* guarda este valor na configuração de autenticação com as *gateways* para localizar a *gateway* com que um cliente se liga.
- **client**: identificador único do cliente. Utilizado para fazer o mapeamento de pacotes e respetivos clientes, garantindo que nenhum cliente tem acesso ao pacote e configuração de outro cliente. O *script* usa este identificador como parte do nome do pacote gerado.
- **features**: lista de funcionalidades do SDK separadas por vírgulas. O *script* usa esta lista para escolher os módulos que vão constituir o SDK a gerar.
- **credentials**: credenciais que permitem a comunicação cliente-*gateway* e *gateway*-servidor SIP, descritas no formato JSON. O *script* guarda estes valores na configuração de autenticação com o servidor SIP.

O *script* tem como base o *script* original para construir o SDK, tendo sido alterado para permitir escolher dinamicamente as funcionalidades do SDK, incluir a aplicação demo para testar o SDK, incluir na aplicação a configuração do cliente (permitindo testar a aplicação sem a necessidade de o cliente adicionar manualmente a configuração) e compactar num pacote que pode ser posteriormente descarregado pelo cliente.

O *script* não faz a validação da configuração, estando esta responsabilidade a cargo do componente que manda executar o *script*, como mencionado anteriormente.

4.1.4 Camada de comunicação

A camada de comunicação é constituída por um conjunto de instâncias da *gateway*, que se podem encontrar dentro ou fora da rede interna da WIT, e o balanceador de carga geográfico Amazon Route 53 (apresentado no **Capítulo 4.1.4.2**).

As instâncias da *gateway* desta camada são versões modificadas que fazem a validação das chaves e das suas permissões recorrendo ao servidor WaaS. As alteração são descritas no **Capítulo 4.1.4.1**.

Para proceder à comunicação, os clientes fazem uso do SDK descarregado na plataforma de gestão para criar uma aplicação de comunicação. Os

clientes começam por fazer um pedido de ligação a uma *gateway* utilizando um domínio comum a todas as instâncias. De forma transparente, o cliente estão a fazer um pedido de DNS ao serviço Amazon Route 53, que foi configurado para distribuir os pedidos pelas instâncias disponíveis com base na localização física do cliente que fez o pedido. Assim, após o pedido de DNS o cliente recebe o IP da instância da gateway disponível que está mais perto dele e faz a autenticação utilizando uma chave de serviço criada na plataforma e a usando a configuração do seu servidor SIP. Após a autenticação bem sucedida o cliente está pronto para iniciar chamadas de voz.

4.1.4.1 Arquitetura da *Gateway*

A *WIT WebRTC Gateway* é constituída por vários módulos que são responsáveis por gerir uma principal, como gerir as configurações, eventos, chamadas, sessões e mensagens. Para este projeto há três módulos que têm uma importância acrescida, os quais tiveram que ser modificados para que a gestão de chaves e chamadas concorrentes passasse a ser feita e sincronizada com o *backoffice* de gestão. Os módulos são os seguintes:

- *API Key Configuration Getter*
- *API Key Manager*
- *Call Manager*

API Key Configuration Getter

O ***API Key Configuration Getter*** é responsável por obter as informações de uma chave da API a partir do seu identificador secreto (*key secret*).

Este módulo foi substituído por uma nova versão que comunica com o *backoffice* para obter a informação de uma chave. A nova implementação usa um cliente HTTP para comunicar com o *backoffice*, faz a interpretação da informação recebida e cria o objeto (com a informação da chave), de acordo com a documentação da *gateway*, para ser tratado pelo sistema.

Os erros devolvidos pelo *backoffice* estão apresentados na API REST para a *gateway* no **Anexo C**.

API Key Manager

O ***API Key Configuration Getter*** é responsável por guardar as chaves associadas a sessões ativas, validar as permissões dos pedidos com base nas permissões das chaves (módulos permitidos, número de sessões e chamadas, limitação da taxa de pedidos e validade).

Este módulo foi substituído por uma nova versão que extende a versão anterior mas que faz a validação dos pedidos de forma ligeiramente diferente. Na nova implementação, quando é recebido um pedido de início de chamada, em vez de ser verificado o limite de chamadas concorrentes com base na informação local, essa validação é feita utilizando um cliente HTTP para comunicar com o *backoffice*, que faz a verificação do limite de chamadas concorrentes de uma chave com base na sua utilização por todas as *gateways*, e no caso de ser permitido, é prontamente reservado um *slot* para essa chave na *gateway* que fez o pedido. No caso de não ser permitido, o erros é identificado com base no código de erro HTTP devolvido, e o módulo devolve um erro de operação não permitida de acordo com a documentação.

Os pedidos de reserva de chamada incluem informação acerca da utilização local dessa chave e um *timestamp*. Assim, no caso de ter havido pedidos para libertação de *slots* que falharam, o *backoffice* obtém informação atualizada acerca da utilização local nessa *gateway* para poder atualizar internamente, usando também o *timestamp* para verificar a informação mais atual.

Os códigos de erro devolvidos pelo *backoffice* estão apresentados na API REST para a *gateway* no **Anexo C**.

Call Manager

O ***Call Manager*** é responsável por iniciar, gerir e terminar as chamadas, enviando notificações e códigos de estado, e criando os canais para comunicação.

Este módulo foi alterado para poder, indiretamente, notificar o *backoffice* quando as chamadas terminam. Assim, quando uma chamada termina e antes da sua informação ser apagada, este módulo notifica o *API Key Manager* de que uma chamada terminou, que por sua vez faz uso do cliente HTTP para

notificar o *backoffice* e libertar um *slot* de chamadas (para a chave utilizada na chamada terminada).

4.1.4.2 Configuração do Amazon Route 53

Como referido anteriormente, o **Amazon Route 53** foi o serviço escolhido como balanceador de carga geográfico. Para utilizar o serviço é necessário possuir um domínio web e é necessário configurá-lo para que utilize os *name servers* disponibilizados pelo Amazon Route 53, e a forma como esta configuração é feita varia com o serviço onde o domínio foi adquirido. Após a configuração dos *name servers*, o Amazon Route 53 passa a responder aos pedidos DNS que são feitos ao domínio configurado.

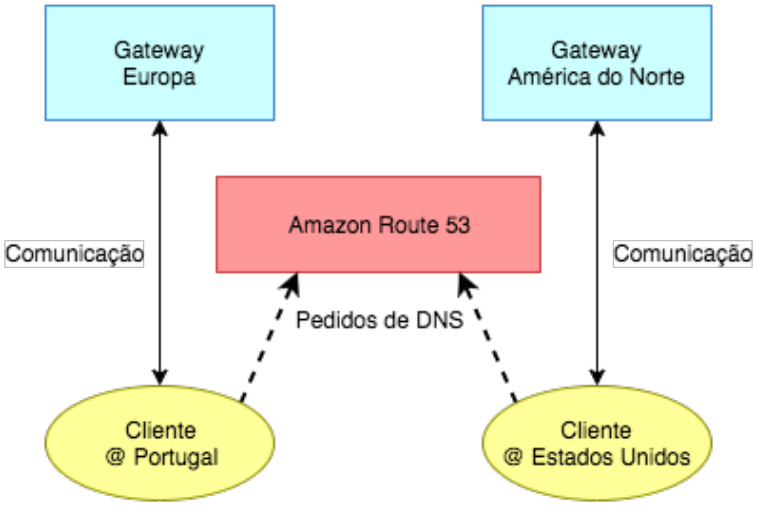
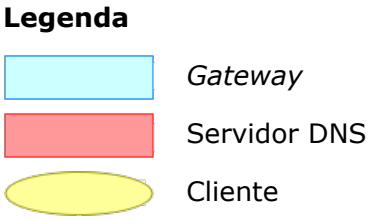


Figura 4.3: Esquema de pedidos de DNS e comunicação no cenário de testes



O serviço pode ser configurado, através da plataforma *web*, com os endereços das máquinas pretendidas. O serviço ignora a localização física das máquinas e requer que sejam explicitamente atribuídas zonas (países ou continente) a cada máquina, e no caso de não ser configurada uma máquina como *default*, os clientes nas zonas sem máquina atribuída ficam sem acesso ao serviço. No caso de haver máquinas atribuídas a um país cujo continente também

tem máquinas atribuídas, o serviço dá prioridade às máquinas da atribuídas ao país por a zona ser mais específica.

Como este projeto era apenas uma prova de conceito, para efeitos de teste foram configuradas apenas duas instâncias com uma região cada: Europa e América do Norte. A *gateway* configurada para servir a Europa também foi configurada como *default*, o que significa que também passa a servir os pedidos dos clientes que não têm o seu IP mapeado a uma localização geográfica, e os clientes de zonas sem *gateway* atribuída (neste caso, clientes fora da Europa e América do Norte).

4.2 Arquitetura da plataforma *web*

A plataforma *web* foi desenvolvida utilizando a *framework AngularJS*. Esta *framework* como principal foco a criação de aplicações *web* de uma única página, constituída por *templates* HTML, modelos de dados e controladores, em que apenas é feita a atualização de partes da página (quando o modelo de dados é alterado ou quando se pretende visualizar um novo componente). Para tornar a aplicação responsiva, ou seja, para que a aplicação se ajustasse a ecrãs de diferentes tamanhos, foi utilizado a biblioteca *Bootstrap*, que já inclui classes que auxiliam essa tarefa.

Nesta secção são inicialmente apresentadas as funcionalidades e os componentes do AngularJS, e a forma como são utilizados no projeto. De seguida são apresentados os componentes da aplicação o o fluxo da navegação pela aplicação.

4.2.1 AngularJS

O *AngularJS* é uma *framework JavaScript* que oferece um modelo MVC extensível que permite separar a aplicação em componentes de lógica e componentes de visualização, e oferece funcionalidades e componentes que permitem tornar a criação da aplicação mais modular.

4.2.1.1 Funcionalidades

As funcionalidades do AngularJS a seguir descritas permitem modularizar a aplicação uma vez que é possível separar a camada de visualização da camada lógica, e interligar os vários componentes da camada lógica sem a necessidade de estes conhecerem as implementações de cada serviço.

Data-binding

O AngularJS oferece uma funcionalidade de mapeamento bidirecional entre a camada de dados e a de visualização. Isto permite sincronizar o conteúdo do modelo de dados e a informação apresentada quando um destes é modificado. Assim, deixa de haver a necessidade de injetar manualmente os valores (do modelo de dados) no código HTML e não há a necessidade de fazer uma pesquisa no DOM para encontrar dados introduzidos pelos utilizadores. Isto permite uma maior separação da vista e do modelo de dados uma vez que os controladores em *JavaScript* não precisam de conhecer o conteúdo da página HTML, limitando-se a oferecer funcionalidade e dados.

No projeto, os pedidos ao servidor são simplesmente para obter os *templates* para visualização e chamar os recursos da API REST para obter o modelo de dados, ficando a apresentação da página a cargo do AngularJS com base no mapeamento entre a visualização e o modelo de dados.

Injeção de dependências

A injeção de dependências é um padrão de desenvolvimento de *software* que permite criar referências para componentes de forma a que as dependências entre componentes não necessitem de ser declaradas explicitamente. Neste padrão existe um injetor que trata de iniciar os componente necessários ao sistema e de os injetar nos componentes que os precisarem. Isto permite criar uma maior independência entre os componentes, uma vez que não necessitam de conhecer os componentes internamente para os criar (uma vez que estes podem ter outras dependências) nem que serviços está realmente a utilizar, apenas precisa de conhecer as interfaces para os poder utilizar. Assim, os serviços podem ser substituídos mais facilmente, e mesmo que tenham diferentes dependências não vai ter impacto nos componentes que os usam

porque a responsabilidade de iniciar as dependências está ao cargo do injetor de dependências.

4.2.1.2 Componentes

Os componentes do AngularJS funcionam como classes que permitem separar grupos de funcionalidades e informações para estruturar melhor a aplicação. O AngularJS permite criar novos componentes dos tipos a seguir apresentados, e também já apresenta um leque de componentes prontos a utilizar, como por exemplo um cliente HTTP, e um router que permite navegar por diferentes páginas (geridas por diferentes controladores) sem a necessidade de atualizar a página completa.

Controladores

Os controladores são componentes que permitem controlar o fluxo de uma aplicação *web* ou de módulos que a constituem. Os controladores armazenam informação que pode ser mapeada a determinados componentes da vista, e disponibilizam métodos que controlam as ações na página, como comunicar com o servidor para aplicar alterações, pedir para atualizar o modelo de dados, e navegar entre as vistas da página.

Diretivas

As diretivas são *templates* HTML de componentes da vista às quais é associado um modelo de dados. Com as diretivas é possível modularizar os componentes da vista para uma melhor organização do código, e reutilizar certos componentes. É ainda possível associar controladores às diretivas, permitindo-lhes gerir a informação que lhes foi associada e o fluxo do seu componente.

Serviços

Os serviços são componentes que permitem complementar os controladores disponibilizando funcionalidades que podem ser utilizadas pelos vários controladores e armazenando dados que podem ser partilhados. Os serviços

são injetados, por referência, nos controladores, ou seja, para cada serviço existe apenas uma instância e esta é partilhada entre todos os controladores que a necessitarem.

4.2.2 Constituição e fluxo da aplicação

A plataforma *web* faz uso do modelo MVC do AngularJS e é portanto constituída por diversos *templates* HTML que constituem os componentes da página, e por controladores responsáveis por comunicar com o servidor para obter o modelo de dados e por controlar o fluxo da aplicação, incluíndo ações a comunicar ao servidor para serem executadas e atualizar a página em função dos pedidos de navegação por parte do utilizador.

A plataforma é constituída por quatro páginas principais, que se encontram logicamente separadas e não formam uma aplicação de página única devido ao seu objetivo e ao seu destinatário. As páginas são as seguintes:

- **Página de gestão do serviço**: permite gerir o serviço (perfis, chaves e contas) e está disponível apenas para utilizadores autenticados.
- **Página de apresentação**: tem como objetivo apresentar o produto a qualquer tipo de utilizador (autenticado e nao autenticado), tem uma menor interação funcional com o servidor (limita-se a apresentar conteúdo de divulgação) e o facto de estar separada da página de gestão permite que ambas sejam geridas de forma independente.
- **Página de registo**: esta página serve apenas para os utilizadores não autenticados criarem uma conta
- **Página de confirmações e validações**: é a página que recebe os pedidos para confirmação de ações ou para fazer validações (como validações de pagamentos). Esta página pode receber pedidos de utilizadores autenticados ou não autenticados (como por exemplo, quando fazem a confirmação do registo), e geralmente, após a validação, reencaminha-os para a página de gestão se estiverem autenticados (no caso da confirmação do registo o servidor autentica automaticamente o utilizador, e no caso da validação de pagamentos são reencaminhados para a página onde podem descarregar o SDK). Esta página é geralmente o resultado de um encaminhamento, a partir de um *email* de confirmação ou a partir do serviço de pagamentos.

A página de gestão do serviço é a que apresenta uma maior complexidade, e apresenta várias vistas sob a forma de uma aplicação de página única. Esta página é descrita de forma mais pormenorizada no **Capítulo 4.2.2.1**. As outras três são geridas por apenas um controlador cada, uma vez que apresentam uma baixa complexidade.

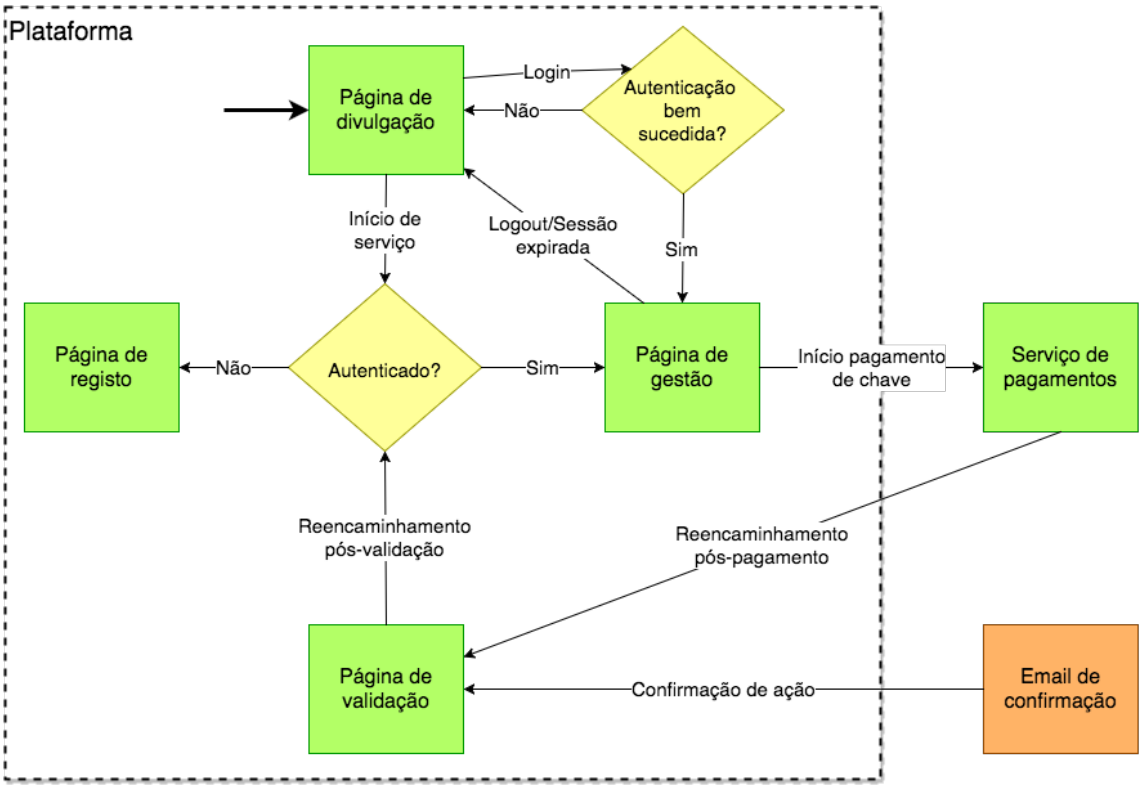
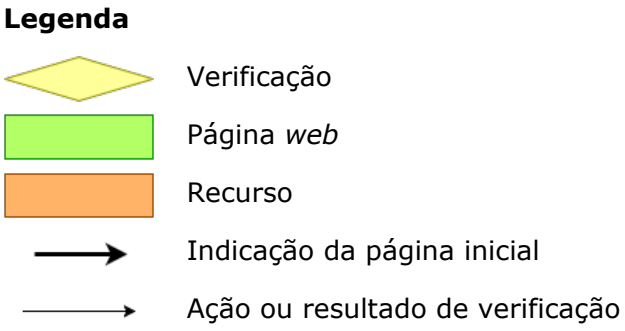


Figura 4.4: Fluxo da navegação pelas páginas da aplicação *web*



Geralmente a primeira página que é apresentada ao utilizador é a de divulgação do serviço (uma vez que é página inicial so serviço) ou a de confirmação de ação (que pode ser aberta a partir de um redirecionamento de um serviço de pagamentos ou a partir de um endereço numa *email* de confirmação).

A partir da página de divulgação o utilizador pode ir para a página de registo e criar uma conta, ou autenticar-se e ir para a página de gestão. Dentro da página de gestão existe um fluxo de navegação próprio (explicado no **Capítulo 4.2.2.1**), e também é possível regressar à página de divulgação (através de uma ligação direta ou sendo redirecionado após terminar a sessão). Na página de sessão também é possível ser encaminhado para uma página de serviço externo de pagamentos, que eventualmente volta a redirecionar o utilizador para a página de gestão (se o pagamento for cancelado) ou para a página de confirmação. Na página de confirmação é apresentada uma mensagem ao utilizador, e após algum tempo este é encaminhado para a página de gestão. Quando um utilizador não autenticado tenta aceder à página de gestão, como não tem autorização para aceder ao serviço é reencaminhado para a página de registo, onde pode criar uma conta ou autenticar-se.

Todas as páginas apresentam o mesmo cabeçalho, que permite fazer a autenticação (ou terminar a sessão, caso já estejam autenticados) e ir para a página principal do serviço (página de divulgação).

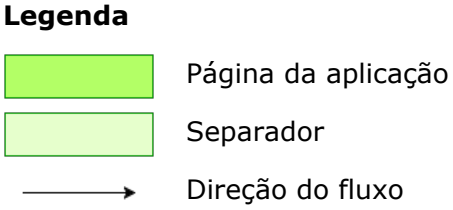
4.2.2.1 Página de gestão

A página de gestão é gerida por vários controladores e serviços. Existem quatro páginas principais, que estão divididas em separadores, e a estrutura é apresentada a seguir:

- **Página de gestão do administrador**: está reservada a utilizadores com permissões de administrador. Inclui dois separadores onde o administrador pode listar as contas registadas e monitorizar a utilização das *gateways*
  - **Lista de contas**: A página de listagem de contas é gerida por um controlador que faz uso do serviço DataRetrievingService para obter os grupos de contas a apresentar. Este controlador apresenta baixa complexidade, uma vez que a tarefa mais exigente (obter as contas) é feita pelo serviço
  - **Monitorização das *gateways***: A página de monitorização é gerida por um controlador que faz uso da biblioteca SockJS-client para criar um *WebSocket* entre a plataforma e o servidor para poder receber, em tempo real, mensagens do servidor e sem a necessidade de fazer

pedidos à rede. O controlador trata das mensagens recebidas e gere o conteúdo que deve ser apresentado

- **Página de gestão do cliente**: Esta página é onde um cliente pode gerir o seu serviço ou onde o administrador pode gerir o serviço de qualquer cliente. A página é constituída por dois separadores: gestão de chaves e gestão do SDK
  - **Lista de chaves**: No separador de gestão de chaves é possível gerar chaves e listar as chaves associadas à conta. O controlador que gere este separador também faz uso do serviço DataRetrievingService para obter as chaves a apresentar
  - **Personalização do SDK**: No separador de gestão do SDK é possível personalizar e descarregar o SDK
- **Página de preferências**: Inclui separadores que permitem atualizar parâmetros da conta
  - **Atualizar dos dados do perfil**: apresenta um formulário para atualizar campos como o nome, empresa e país
  - **Alteração da *password***: apresenta um formulário para substituir a *password*
  - **Alteração do *email***: apresenta um formulário para substituir o *email* associado à conta
- **Página da chave**: permite ver as configurações da chave, e iniciar o pagamento de chaves por pagar
  - **Detalhes da chave**: apresenta os detalhes de uma chave e permite encaminhar o utilizador para a página do serviço de pagamentos, caso a chave ainda esteja por pagar
  - **Partilha da chave**: permite partilhar o *key secret* da chave (que permite autenticar com as *gateways*) com adicionando endereços de *email* para onde deve ser enviada



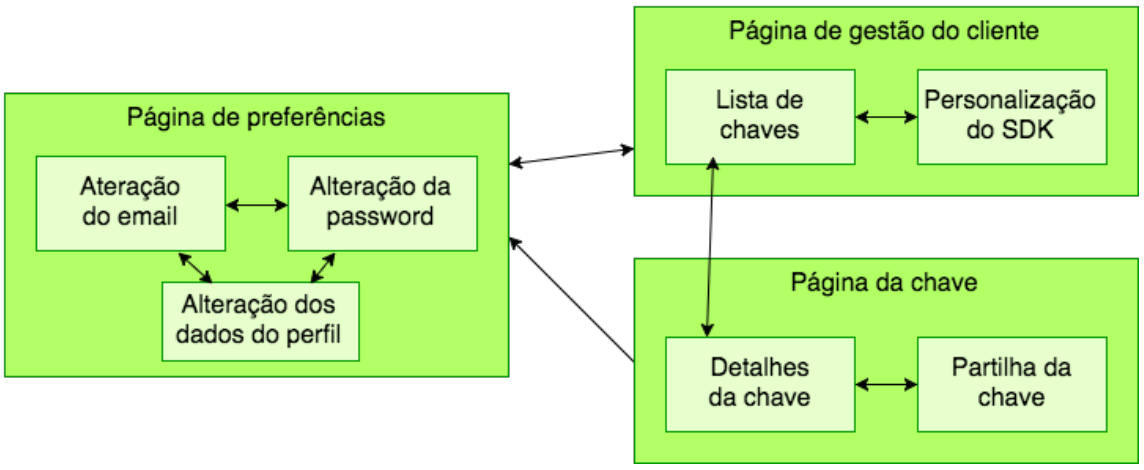


Figura 4.5: Fluxo da navegação na página de gestão, na versão de cliente

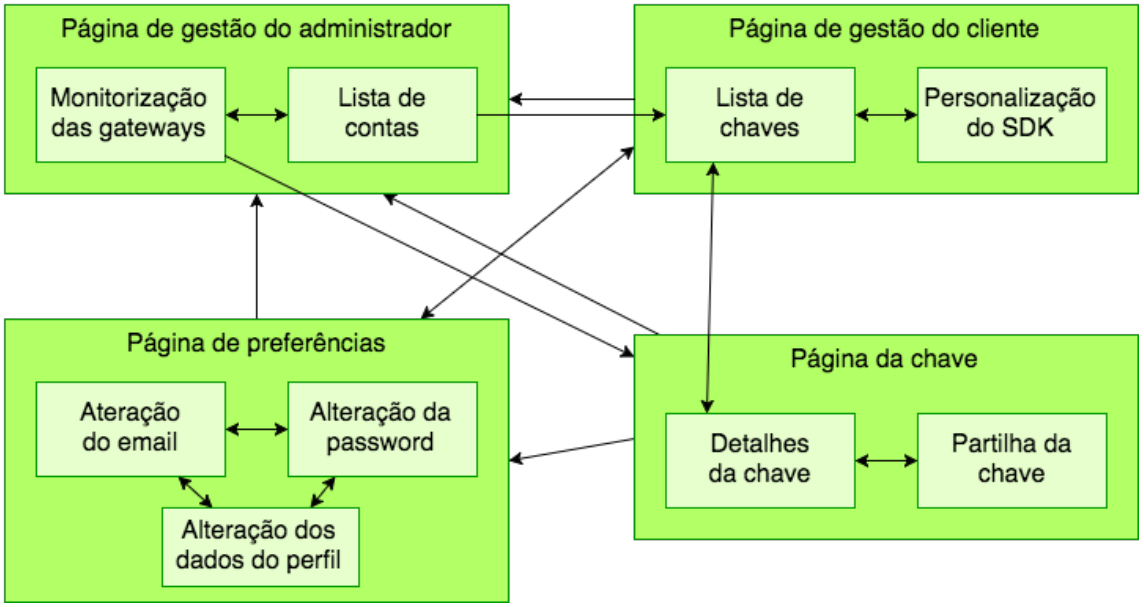


Figura 4.6: Fluxo da navegação na página de gestão, na versão de administrador

Componentes

A página é constituída por vários componentes recicláveis que permitem reduzir a complexidade do projeto, como por exemplo: janelas de diálogo, botões de paginação e *dropdowns*. Estes componentes incluem o conteúdo da visualização e um controlador, sendo apenas necessário atribuir-lhes o modelo de dados a apresentar.

Serviços

Os serviços a seguir apresentados foram criados para tratar de tarefas que se repetiam e eram necessárias nos vários controladores.

- **DataRetrievingService**: permite lidar com a necessidade de obter recursos com uma quantidade de informação que se pode tornar elevada e que obriga a que esta seja distribuída por várias respostas. Este serviço funciona como cache dessa informação, o que permite aos controladores pedirem a informação a apresentar sem ter que repetir os pedidos ao servidor.
- **PaginationService**: permite gerir a lista de botões a apresentar para alterar o número da página, com base no número da página atual, número de elementos apresentados e número de elementos total.
- **DialogService**: permite apresentar uma caixa de diálogo com mensagens informativas ou de erro, necessitando apenas de atribuir o título e mensagem.

Capítulo 5

Testes

Neste capítulo é apresentada a lista de testes funcionais realizados às funcionalidades da plataforma *web*, o que permitiu também verificar se o *backoffice* também apresentava o comportamento esperado. Esta lista é apresentada de forma mais detalhada no **Anexo G**. Na segunda secção deste capítulo é feita uma referência às métricas dos testes de carga, no entanto não foi possível chegar a testá-las.

5.1 Testes funcionais

Os testes funcionais foram sendo definidos e realizados ao longo do desenvolvimento, quando eram adicionadas novas funcionalidades, e foram novamente testados no fim do processo de desenvolvimento.

Os testes TF.3.1 a TF.5.2 descrevem funcionalidades que podem ser executadas na plataforma de gestão, sendo restritas a utilizadores autenticados. Estes pedidos servem para consultar informação referente a uma conta e gerir o seu serviço, e como tal o servidor var a validação de todos os pedidos para garantir que os utilizadores têm permissões para consultar e modificar o serviço. Estas permissões garantem que um cliente apenas tenha acesso à sua conta, mas um administrador tem acesso às contas de todos os utilizadores que não tenham permissões de administrador, e os administradores não têm acesso às contas de outros administradores. Este conjunto de testes começou por ser executado através de uma conta de cliente para gerir o seu próprio serviço.

O teste TF.6.5 demonstra que os administradores têm acesso à conta de clientes e o teste TF.6.6 demonstra que os administradores não têm acesso à conta de outros administradores. Com estes dois testes aprovados, os testes TF.3.1 a TF.5.2 foram novamente executados através de uma conta de administrador a gerir uma conta de utilizador e foram igualmente aprovados.

Registo	
ID	**Teste**
T.1.1	Registo bem sucedido
T.1.2	Formulário de registo incompleto
T.1.3	Endereço de email inválido
T.1.4	Password com tamanho inválido
T.1.5	Password não cumpre os requisitos mínimos de segurança
T.1.6	Password de confirmação inválida
T.1.7	Email em uso por outra conta
T.1.8	Confirmação de registo bem sucedida
T.1.9	Cancelamento de registo bem sucedido
T.1.10	Confirmação ou cancelamento de registo com código expirado
T.1.11	Confirmação ou cancelamento de registo com código inválido

Login	
ID	**Teste**
T.2.1	Login com credenciais válidas
T.2.2	Login com credenciais inválidas
T.2.3	Login com conta não confirmada
T.2.4	Login com conta bloqueada
T.2.5	Reenviar email de confirmação de conta

Chaves	
ID	**Teste**
T.3.1	Conta sem chaves
T.3.2	Criação de chave com sucesso
T.3.3	Limite de chaves inativas
T.3.4	Listagem de chaves
T.3.5	Listagem paginada de chaves
T.3.6	Navegação pela lista de chaves
T.3.7	Listagem de chaves após redimensionamento da janela do browser
T.3.8	Listagem de chaves por estado
T.3.9	Listagem de chaves por estado sem resultados
T.3.10	Início do processo de pagamento com sucesso
T.3.11	Serviço de pagamentos indisponível
T.3.12	Confirmação do pagamento com sucesso
T.3.13	Confirmação do pagamento sem sucesso

SDK	
ID	**Teste**
T.4.1	Obtenção de chaves ativas com sucesso
T.4.2	Obtenção de chaves ativas sem sucesso
T.4.3	Geração e download do SDK com sucesso
T.4.4	Configuração inválida na geração do SDK
T.4.5	Geração do SDK indisponível

Autenticação	
ID	**Teste**
T.5.1	Sessão expirada
T.5.2	Pedido não autorizado

Contas	
ID	**Teste**
T.6.1	Listagem de contas
T.6.2	Listagem paginada de contas
T.6.3	Navegação pela lista de contas
T.6.4	Listagem de contas após redimensionamento da janela do browser
T.6.5	Configuração do serviço de um cliente
T.6.6	Acesso negado às contas de outros administradores

Comunicação	
ID	**Teste**
T.7.1	Fazer a autenticação com a gateway WebRTC usando a aplicação demo
T.7.2	Autenticação com a gateway WebRTC usando a aplicação demo e uma chave inativa
T.7.3	Autenticação com a gateway WebRTC usando a aplicação demo e uma chave expirada
T.7.4	Autenticação com a gateway WebRTC usando a aplicação demo e configuração inválida do servidor SIP
T.7.5	Autenticação com a gateway WebRTC usando a aplicação demo e configuração inválida do servidor SIP
T.7.6	Chamada de voz para um telemóvel, usando a aplicação demo
T.7.7	Chamada de voz para um telemóvel desligada após ser atendida, usando a aplicação demo
T.7.8	Chamada de voz para um telemóvel desligada após ser atendida, usando a aplicação demo
T.7.9	Autenticação com a gateway WebRTC usando um snippet
T.7.10	Chamada de voz para um telemóvel através da gateway WebRTC, usando um snippet
T.7.11	Tentativa de chamada de voz utilizando uma chave com o limite de chamadas concorrentes atingido

Os testes TF.7.9 e TF.7.10 são executados usando apenas o SDK e um

snippet com funcionalidade básica de autenticação e chamadas de voz através da *gateway*, que pretende demonstrar a forma com o SDK facilita a criação de uma aplicação funcional de comunicação. O snippet é constituído por 3 blocos (configuração, autenticação e chamada de voz), não tem tratamento de erros e serve apenas como demonstração, sendo apresentado na página de divulgação do serviço.

5.2 Testes de carga

Os testes de carga permitem obter informação acerca de quanto da utilização máxima que o servido é capaz de processar, o que ajuda a saber até onde é possível escalar o sistema e quais os pontos mais críticos.

Nestes testes as métricas eram:

- Número máximo de chamadas concorrentes por *gateway*
- Número máximo de pedidos por segundo recebidos pelo *backoffice*
- Número máximo de instâncias da *gateway* a comunicar com o *backoffice*

Os testes de carga não chegaram a ser feitos devido à falta de tempo causada pela complexidade das tarefas de configuração e alteração das gateways. No entanto, o teste da terceira métrica não iria ser possível de realizar, uma vez que não tinha condições de ter um cenário com um número suficiente de instâncias para avaliar essa métrica.

Capítulo 6

Conclusão

Neste capítulo é feita uma reflexão sobre trabalho desenvolvido ao longo do estágio e são feitas algumas sugestões para um trabalho futuro.

Neste projeto foi criado de raiz um serviço que permite gerir a comercialização de um serviço de comunicação da WIT Software sem haver a necessidade da sua intervenção. Este serviço de gestão é constituído por um *backoffice* que gere a informação do serviço, e por uma plataforma à qual os clientes acedem para subscrever o serviço. O produto do serviço de comunicação foi alterado para permitir a gestão de forma centralizada, e foi alterada a abordagem de distribuição do serviço: em vez de ser comercializada uma *gateway* é disponibilizado um serviço de *cloud Software as a Service* que os clientes podem subscrever.

No primeiro semestre, após o estudo dos produtos concorrente e da análise das tecnologias, foram criadas as bases para a plataforma e para o servidor de gestão, e foram implementadas as funcionalidades de registo e geração chaves do serviço.

O segundo semestre foi mais dedicado ao desenvolvimento. No serviço de gestão foram integrados os serviços de pagamentos e de balanceamento de carga, foi implementada a funcionalidade de monitorização da utilização das *gateways*, foi implementada a UI, foi integrada a geração do SDK para permitir a geração do SDK personalizado (com os módulos pretendidos e com a configuração do servidor SIP dos clientes) de forma dinâmica e foi criada a página de divulgação do produto. Na parte de comunicação, foi alterada a *gateway* para comunicar com o serviço de gestão para obter a informação das chaves do serviço e para fazer a contagem de chamadas concorrentes.

Este projeto permitiu aprender uma metodologia de desenvolvimento num ambiente profissional e permitiu aumentar os conhecimentos de desenvolvimento de *software* ao dar a conhecer e permitir utilizar novas *frameworks* de desenvolvimento para *backend* e *frontend*. O estudo do projeto da *gateway* ajudou a conhecer boas práticas de estruturação dos componentes num projeto, facilitando o desenvolvimento e a possível futura extensão.

Relativamente aos objetivos pretendidos, o protótipo cumpriu os objetivos inicialmente propostos: permite subscrever o serviço ao criar chaves de serviço e efetuar pagamentos pelo PayPal, permite utilizar as chaves criadas para autenticar com a *gateway*, permite manter os dados da utilização das chaves centralizada no servidor de gestão e manter as restrições do serviço subscrito, permite distribuir os cliente pelas *gateways* com base na localização física dos clientes, e permite à plataforma *web* fazer a gestão do serviço com recurso a uma API REST criada para o efeito. No entanto não foi possível chegar a testar e avaliar o projeto do ponto de vista de carga suportada.

6.1 Trabalho futuro

A prova de conceito desenvolvida tem potencial para serem adicionadas novas funcionalidades, de forma a oferecer um melhor serviço ao cliente.

Como referido no **Capítulo 2.2.4**, a plataforma não precisa de estar limitada a disponibilizar um único serviço de pagamentos. O servidor de gestão foi preparado para que possam ser integrados novos serviços de pagamentos, e no futuro seria possível integrar outros serviços como o Google Wallet.

Outras funcionalidades que podem ser consideradas para uma extensão futura do protótipo desenvolvido são:

- **Suporte de alertas**: enviar *emails* aos cliente quando as chaves do serviço estão para expirar, quando expiram e quando o limite de chamadas concorrentes é atingido, e notificar o administrador quando são efetuados pagamentos.
- ***Logs* de utilização das chaves**: aceder aos *logs* gerados pelas *gateways* para apresentar aos clientes e administradores informação acerca da utilização das chaves, com a possibilidade de gerar gráficos e relatórios.

- **Faturação**: permitir gerar informação de faturação após os pagamentos.
- ***Dashboard* de administrador**: apresentar gráficos com a utilização do serviço, nomeadamente visualização do número de registos e acessos ao longo do tempo, pagamentos e estatísticas de utilização.

Outra tarefa a realizar no futuro seriam os testes de carga de forma a avaliar os pontos críticos do sistema, do ponto de vista da escalabilidade. Após estes testes podia ainda haver a necessidade de proceder a alterações na arquitetura da camada de comunicação, na forma como é feita a centralização da gestão de chaves ou proceder a otimizações para reduzir os pedidos ao *backoffice* para a validação das chaves.

Bibliografia

[1] Demystifying WebRTC, André Silva, 2014.

[2] WIT WebRTC Gateway. https://www.wit-software.com/products/webrtc/. [Online; acedido a 2 de Outubro de 2015].

[3] Scrum Workflow. https://www.emaze.com/@AZLZFOCI/Scrum-Workflow. [Online; acedido a 4 de Agosto de 2016].

[4] WebRTC Peer-to-peer connections. http://caniuse.com/#feat=rtcpeerconnection, 2016. [Online; acedido a 2 de Agosto de 2016].

[5] Web Sockets. http://caniuse.com/#feat=websockets, 2016. [Online; acedido a 2 de Agosto de 2016].

[6] PayPal. https://www.paypal.com/uk/webapps/mpp/home. [Online; acedido a 2 de Agosto de 2016].

[7] ABC WebRTC Gateway. http://www.frafos.com/webrtc/. [Online; acedido a 02 de Novembro de 2015].

[8] Ahoy Gateway. https://ahoyrtc.com/portfolio-en/ahoygateway.html. [Online; acedido a 05 de Novembro de 2015].

[9] Amazon Route 53 Pricing. https://aws.amazon.com/pt/route53/pricing. [Online; acedido a 11 de Janeiro de 2016].

[10] Amazon Route 53. https://aws.amazon.com/pt/route53. [Online; acedido a 11 de Janeiro de 2016].

[11] AngularJS. https://en.wikipedia.org/wiki/AngularJS. [Online; acedido a 05 de Novembro de 2015].

[12] Apache Tomcat. http://tomcat.apache.org/. [Online; acedido a 05 de Novembro de 2015].

[13] BIND – The most widely used Name Server Software. https://www.isc.org/downloads/bind. [Online; acedido a 11 de Janeiro de 2016].

[14] Choosing a front end framework: Angular vs. Ember vs. React. http://smashingboxes.com/blog/choosing-a-front-end-framework-angular-ember-react. [Online; acedido a 11 de Janeiro de 2016].

[15] GeoDNS BIND patch. http://www.caraytech.com/geodns. [Online; acedido a 11 de Janeiro de 2016].

[16] Getting Started with Plivo. https://www.plivo.com/docs/getting-started/. [Online; acedido a 02 de Novembro de 2015].

[17] Google Cloud DNS Pricing. https://cloud.google.com/dns/pricing. [Online; acedido a 11 de Janeiro de 2016].

[18] Google Cloud DNS. https://cloud.google.com/dns/docs. [Online; acedido a 11 de Janeiro de 2016].

[19] Google Wallet. https://www.google.com/wallet/. [Online; acedido a 11 de Janeiro de 2016].

[20] Justin Uberti and Sam Dutton. 2013. WebRTC - Plugin-free realtime communication. http://io13webrtc.appspot.com/. [Online; acedido a 16 de Outubro de 2015].

[21] Multiple Levels of Done. https://www.mountaingoatsoftware.com/blog/multiple-levels-of-done. [Online; acedido a 22 de Outubro de 2015].

[22] MySQL. https://en.wikipedia.org/wiki/MySQL. [Online; acedido a 11 de Janeiro de 2016].

[23] Oracle Communications WebRTC Session Controller. http://www.oracle.com/us/products/applications/communications/web-rtc-session-controller/. [Online; acedido a 02 de Novembro de 2015].

[24] Oracle Database. https://en.wikipedia.org/wiki/Oracle_Database. [Online; acedido a 11 de Janeiro de 2016].

[25] PostgreSQL. https://en.wikipedia.org/wiki/PostgreSQL. [Online; acedido a 05 de Novembro de 2016].

[26] React (JavaScript library). https://en.wikipedia.org/wiki/React_(JavaScript_library). [Online; acedido a 11 de Janeiro de 2016].

[27] SPiDR™ WebRTC Gateway. http://www.genband.com/products/experius/webrtc-gateway. [Online; acedido a 02 de Novembro de 2015].

[28] Spring Framework. https://en.wikipedia.org/wiki/Spring_Framework. [Online; acedido a 10 de Novembro de 2015].

[29] SQLite. https://en.wikipedia.org/wiki/SQLite. [Online; acedido a 11 de Janeiro de 2015].

[30] The 10 Most Popular Online Payment Solutions. https://www.searchenginejournal.com/the-10-most-popular-online-payment-solutions/. [Online; acedido a 11 de Janeiro de 2016].

[31] The Spring IO Platform . https://spring.io/platform. [Online; acedido a 10 de Novembro de 2015].

[32] The WebSocket Protocol. https://tools.ietf.org/html/rfc6455. [Online; acedido a 16 de Outubro de 2015].

[33] Twilio WebRTC. https://www.twilio.com/webrtc. [Online; acedido a 02 de Novembro de 2015].

[34] WebRTC Gateway. https://www.wit-software.com/products/webrtc/. [Online; acedido a 05 de Novembro de 2015].

[35] WebRTC Services Solution. http://www.sonus.net/en/products/webrtc/webrtc-services-solution. [Online; acedido a 02 de Novembro de 2015].

[36] WebRTC. http://www.webrtc.org/. [Online; acedido a 16 de Outubro de 2015].

[37] WebRTC. https://en.wikipedia.org/wiki/WebRTC. [Online; acedido a 16 de Outubro de 2015].

[38] WebSocket. https://en.wikipedia.org/wiki/WebSocket. [Online; acedido a 16 de Outubro de 2015].