FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

IPBrick - API para integração de aplicações “third party”

João Lopes Ferreira Sobreira

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: João Manuel Couto das Neves

Co-orientador: Miguel Ramalhão Ribeiro

11 de Agosto de 2014

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2013/2014

A Dissertação intitulada

"IPBrick - API para Integração de Aplicações "Third Party""

foi aprovada em provas realizadas em 16-07-2014

o júri

Presidente Professor Doutor Paulo José Lopes Machado Portugal
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor José Carlos Leite Ramalho
Professor Auxiliar do Departamento de Informática da Escola de Engenharia da Universidade do Minho

Professor Doutor João Manuel Couto das Neves
Professor Auxiliar Convidado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - João Lopes Ferreira Sobreira

Faculdade de Engenharia da Universidade do Porto

Resumo

No âmbito da empresa IPBrick S.A., o projeto da presente dissertação visa a criação de um conjunto de ferramentas disponibilizadas através de uma *Application Programming Interface* (API) para integração de forma independente mas controlada de aplicações *third party* numa plataforma IPBrick. Neste documento é apresentado e descrito o contexto proposto de criação e integração da API a desenvolver, feita uma análise às opções de implementação disponíveis, sendo o foco colocado entre a manutenção da solução original da API de comunicação baseada no protocolo *Simple Object Access Protocol* (SOAP) com definição de serviços em *Web Service Description Language* (WSDL) em contraste com a adoção de uma arquitetura mais recente baseada em *Representational State Transfer* (REST) com mensagens transmitidas em *JavaScript Object Notation* (JSON) e explicado todo o processo de desenvolvimento, decisões tomadas e metodologias utilizadas no decorrer do projeto. O documento termina com uma demonstração de funcionamento do produto final, apresentação de resultados dos testes de validação elaborados e integração real de uma aplicação de teste e de aplicações IPBrick disponibilizadas comercialmente.

Abstract

Under the company IPBrick S.A., this dissertation project aims for the creation of a set of tools available through an *Application Programming Interface* (API) in order to provide an independent but controled third party applications integration process in a IPBrick platform. In this document is presented and described the context in which the aforementioned API will be created and integrated, made an analysis of all available implementation options with emphasis in maintaining the original communication API basis in Simple Object Access Protocol (SOAP) with service definition in Web Service Description Language (WSDL) in contrast to the adoption of a newer architecture based on Representational State Transfer (REST) with messages transmitted in JavaScript Object Notation (JSON) as well as explained the whole development and decision-making process and methodologies used throughout the project. The document ends with a functional demonstration of the final product, presentation of the developed validation tests results and integration of one test application and two IPBrick applications.

Agradecimentos

Apesar de consistir num trabalho individual seria sem dúvida muito mais complicado sem a intervenção de um conjunto de pessoas a quem gostaria aqui de agradecer:

Ao Professor Doutor João Manuel Couto das Neves agradeço por todas as semanas me ter alertado para muitas situações que me passariam despercebidas e me ter guiado e ajudado a tomar grande parte das decisões que enfrentei ao longo do projeto.

Ao Engenheiro Miguel Ramalhão Ribeiro agradeço por me providenciar todas as condições de trabalho possíveis e me incentivar a procurar fazer sempre mais e melhor ao longo do meu trabalho.

A todos os colaboradores das empresas IPBrick SA e iPortalMais, em particular ao Engenheiro Hélder Santos, agradeço pelo tempo dispensado durante o horário de trabalho para me providenciar todo o apoio que fui necessitando.

Ao meu colega Luis Borges que me acompanhou nesta estadia na IPBrick SA agradeço por estar presente para discutir vários aspetos do meu trabalho no decorrer do projeto.

Aos meus familiares e à minha namorada agradeço por todo o apoio moral providenciado durante esta fase que me permitiu focar-me e empenhar-me no trabalho que tinha de realizar.

João Lopes Ferreira Sobreira

"Bernard of Chartres used to compare us to [puny] dwarfs perched on the shoulders of giants. He pointed out that we see more and farther than our predecessors, not because we have keener vision or greater height, but because we are lifted up and borne aloft on their gigantic stature."

John of Salisbury

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos e Abreviaturas

AMF	Action Message Format
API	Application Programming Interface
CNAME	Canonical Name
CPA	Choosen-Plaintext Attack
CPU	Central Processing Unit
CRM	Customer Relationship Management
CRUD	Create Read Update Delete
CSV	Comma-Separated Values
DCOM	Distributed Component Object Model
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FTP	File Transfer Protocol
GIOP	General Inter-ORB Protocol
HATEOAS	Hypermedia As The Engine Of Application State
HMAC	Hash-based Message Authentication Code
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDS	Intrusion Detection System
IIOP	Internet Inter-ORB Protocol
IM	Instant Messaging
IP	Internet Protocol
JMS	Java Message Service
JSON	JavaScript Object Notation

KML	Keyhole Markup Language
LDAP	Lightweight Directory Access Protocol
MX	Mail Exchange
PBX	Private Branch Exchange
PHP	PHP Hypertext Preprocessor
POP	Post Office Protocol
RDF	Resource Description Framework
REST	Representational State Transfer
RPC	Remote Procedure Call
RSS	Rich Site Summary
SAML	Security Assertion Markup Language
SMS	Short Message Service
SMTP	Simple Mail Transport Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UCoIP	Unified Communications over IP
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice Over IP
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WS-I	Web Services Interoperability
XML	Extensible Markup Language
XSD	XML Schema
XSS	Cross Site Scripting

Capítulo 1

Introdução

1.1 Motivação

Num mundo em que cada vez mais serviços estão disponíveis e a sua complexidade se torna cada vez maior é então necessária e vantajosa a cooperação entre entidades especializadas em áreas distintas. Esta cooperação prevê uma fase de integração, sendo portanto crucial que esta seja célere e eficaz de maneira a tornar o serviço funcional no menor intervalo de tempo possível.
Sendo a IPBrick uma solução tecnológica de prestação de serviços de comunicação e de gestão de redes privadas de computadores entre outros, é comum o interesse de aplicações de terceiros na utilização destes serviços em funcionalidades disponibilizadas. A falta de ferramentas que permitam entidades externas desenvolverem as suas aplicações independentemente com certeza de compatibilidade com a solução IPBrick leva a uma necessidade de intervenção da equipa de desenvolvimento da IPBrick S.A. nesta fase de integração.

1.2 Estrutura da Dissertação

No primeiro capítulo da presente dissertação é apresentado e contextualizado o problema do projeto, sendo definidos e enumerados os objetivos propostos aquando a sua abordagem e apresentado o sistema sobre qual é pretendido que a solução seja desenvolvida.
No capítulo 2 são descritas, analisadas e comparadas as tecnologias, soluções e modelos identificados como necessários na abordagem ao problema proposto, sendo apresentadas estatísticas, teorias e testes sempre que haja necessidade de ser efetuada um escolha.
O capítulo 3 apresenta todo o planeamento da solução desenvolvida assim como o método de implementação, estrutura e funcionalidades da mesma. São apresentados resultados de validação da solução e é demonstrado o seu correto funcionamento no capítulo 4 sendo, por fim no capítulo 5, feita uma análise sobre o trabalho desenvolvido e a solução final apresentada sendo também apresentadas propostas para trabalhos futuros no âmbito do projeto.

1.3 IPBrick

O sistema IPBrick é uma solução tecnológica integrada de comunicações, segurança e armazenamento de dados entre outros serviços, de fácil e rápida implementação, criada em Portugal e baseada em *software open-source* Linux. Graças à existência de uma interface gráfica *web* é possível efetuar a gestão de uma rede e dos seus servidores sem conhecimentos avançados das tecnologias utilizadas.
A plataforma é pioneira no conceito Unified Communications over IP (UCoIP) fornecendo a comodidade da utilização de uma vasta gama de serviços de comunicação em rede (e-mail, Voice Over IP (VoIP), Short Message Service (SMS) e Instant Messaging (IM) entre outros) com apenas um único endereço. É estruturada em módulos, com cada módulo focado num aspeto da dinâmica de uma rede descritos nas secções 1.3.1 e 1.3.2.

1.3.1 IPBrick.I

A IPBrick.I é o módulo IPBrick que fornece ferramentas de gestão de redes intranet e dos seus servidores tais como servidor de e-mail, de ficheiros, domínio, impressão, fax e de *backup*. Pode ser utilizado em três modos: *Master, Slave* e *Client*.

IPBrick *Master*
Modo *default* no qual todos os serviços usam o servidor Lightweight Directory Access Protocol (LDAP).

IPBrick *Slave*
Neste modo o servidor *Slave* deverá manter uma cópia sincronizada do servidor *Master* indicado. Permite a autenticação mas não a gestão do LDAP.

IPBrick *Client*
A autenticação neste servidor é feita remotamente, no servidor LDAP, não sendo guardada localmente informação relativa a este processo. É comum a utilização deste modo em máquinas de *relay* como *proxies* e *firewalls*.

Fornece o seguinte conjunto de serviços:

- Servidor de áreas de trabalho individuais e de grupo
- Servidor de gestão de directório *active directory*
- Servidor de imagens de estações de trabalho
- Servidor de Domínio (com *roaming-profiles* e *centralized scripting*)
- Servidor de Dynamic Host Configuration Protocol (DHCP)
- Servidor de Domain Name System (DNS)
- Servidor de correio eletrónico

Serviço de Anti-SPAM
Serviço de Antivírus para correio-eletrónico e área de trabalho
Servidor de impressoras
Servidor de autenticação centralizada
Servidor de base de dados
Servidor de *Groupware* (Calendário e Livro de Endereços)
Servidor de *Backup*

Este módulo também permite a incorporação de aplicações *third party* como o sistema de gestão documental iPortalDoc, o software de *backup* Bacula, o servidor de Customer Relationship Management (CRM) SugarCRM e o sistema de monitorização de redes Nagios entre outros.

1.3.2 IPBrick.C

Com o módulo IPBrick.C é possível equipar uma rede com mecanismos de comunicação e gestão de fluxo de e para a Internet como o serviço de correio eletrónico, VoIP, *proxy* HTTP e FTP e *firewall*.
Fornece o seguinte conjunto de serviços:

Relay de correio eletrónico
Webmail
Servidor *web*
Servidor *Proxy* (HTTP, HTTPS, FTP com estatísticas)
Traffic shaping
Segurança baseada em pacotes
Intrusion Detection System (IDS)
Servidor de VoIP
Integração transparente com Private Branch Exchange (PBX) (ISDN E1/BRI e linhas analógicas)
Serviço *Mail 2 SMS*

1.4 Objetivos

É a intenção deste projeto a revisão e extensão da Application Programming Interface (API) existente no sistema IPBrick no que toca a integração de aplicações *third party*. Tendo em consideração a existência de uma API elaborada mas não completa, uma revisão ajudará a identificar a base sobre a qual as funcionalidades previstas poderão ser desenvolvidas como extensão da API atual. Atualmente a integração de uma aplicação *third party* no sistema é efetuada exclusivamente pela equipa IPBrick pois existe a necessidade de incluir comandos de acesso direto ao sistema nos pacotes a elaborar. Esta situação levanta dois problemas:

1. Nenhuma aplicação destinada ao sistema IPBrick pode ser desenvolvida isoladamente por uma entidade externa;

2. Decorre uma ocupação de recursos por parte da IPBrick no desenvolvimento da integração da aplicação na plataforma.

Surge então a necessidade do desenvolvimento de ferramentas para que entidades externas o possam fazer de maneira independente com certeza de que existe compatibilidade, comunicação e integração do seu produto e assegurando o acesso seguro e controlado ao sistema.

1.5 Análise de Requisitos

O desenvolvimento de um produto exige a identificação de um conjunto de parâmetros e funcionalidades de implementação necessária e prioritária. Após discussão e análise do problema em mãos em conjunto com a equipa IPBrick foram definidas não só funcionalidades requisitadas aquando a integração de uma aplicação num sistema IPBrick mas também caraterísticas de grande interesse para a empresa no que toca à inclusão deste módulo no seu sistema. Optou-se então pela definição de um conjunto de requisitos da solução a desenvolver numa tabela completa e detalhada apresentada no anexo A.1 aqui abordada e explicada por partes.

Tabela 1.1: Tabela de requisitos gerais

G	Geral
1	**Identificação da origem do pedido**
2	**Autenticação dos pedidos**
3	**Verificação da ligação à base de dados**
4	**Apresentação de erros com mensagens informativas quanto à sua origem**
5	**Registos de acesso em ficheiro de texto**
6	**Registos de erros em ficheiro de texto**

Começando por analisar os requisitos de cada acesso à API a desenvolver (tabela 1.1), torna-se necessário garantir a identificação da origem do pedido para efeitos de controlo de acesso e elaboração de registos mais detalhados sendo ainda de grande importância providenciar outros métodos de segurança. Tendo em conta a constante utilização das bases de dados IPBrick, é imperial a verificação do estabelecimento de uma ligação aquando a chegada de um pedido e antes da tentativa de execução das ações pretendidas sendo preferencialmente, no caso de pedidos mal-sucedidos, retornadas respostas com informação relevante para a identificação da falha originária. Por fim apresenta-se como boa prática e de grande utilidade para posterior depuração de erros o registo de todos os acessos efetuados à API e de todas as mensagens de erro reportadas.

Tabela 1.2: Tabela de requisitos de instalação

A	Processo de Instalação
1	**Estado da Instalação**
1.1	Identificação do estado da instalação ("a decorrer"/ "não inicializada")
2	**Início da Instalação**
2.1	Validação de dados
2.1.1	Verificação de compatibilidade da versão IPBrick
2.1.2	Verificação de dupla instalação

2.1.3	Verificação de dependências
2.1.4	Verificação de conflitos
2.2	Activação de flag de instalação
3	**Finalização / Cancelamento da Instalação**
3.1	Desactivação da flag de instalação

Abordando agora serviços a serem disponibilizados pelo produto apresenta-se como principal requisito da solução a desenvolver a automatização de um conjunto de métodos de verificação do estado da aplicação e do sistema aquando a sua instalação como verificação de dependências, conflitos e validação de dados entre outros (tabela 1.2) tendo sido para isso desenvolvido um serviço de controlo do processo de instalação. Este serviço permite também, com a estabilidade do sistema em vista, garantir a não-ocorrência de problemas de concorrência ao implementar um estado de instalação exclusivo. As chamadas a este serviço tornam-se então de caráter obrigatório no inicio de todos os processos de instalação.

Tabela 1.3: Tabela de requisitos do serviço de gestão de utilizadores

B	**Gestão de Utilizadores**
1	**Informação sobre utilizador**
2	**Criação de utilizador**
2.1	Criação de conta de e-mail
3	**Alteração de dados de utilizador**
4	**Eliminação de utilizador**

No que toca a serviços oferecidos pelo sistema IPBrick, foi identificada a necessidade de garantir a gestão de contas LDAP (tabela 1.3) assim como configurações de serviços de comunicação associados (correio eletrónico, endereço VoIP).

Tabela 1.4: Tabela de requisitos do serviço de gestão de regras *firewall*

C	**Gestão de *Firewall***
1	**Informação sobre regra(s) na *firewall***
1.1	Filtragem por parâmetros (protocolo / porta / IP de destino / IP de origem)
2	**Adição de regra na *firewall***
3	**Eliminação de um regra na *firewall***

Outro ponto de interesse de uma aplicação com necessidade de utilização de recursos de rede prende-se com o controlo de regras de *firewal* (tabela 1.4), permitindo a adição da sua própria exceção e identificação de regras com um conjunto de configurações definidas.

Tabela 1.5: Tabela de requisitos do serviço de gestão de *virtual hosts*

D	**Gestão de *Virtual Hosts***
1	**Informação sobre *virtual vost***
1.1	identificação dos parâmetros de configuração associados ao *virtual host*
1.2	Identificação das contas FTP associadas a um *virtual host*
2	**Criação de *virtual host***

2.1	Definição de parâmetros de configuração do *virtual host*
2.2	Adição de registo CNAME
2.3	Criação de conta FTP
3	**Modificação do *virtual host***
3.1	Modificação dos parâmetros de configuração do *virtual host*
3.2	Modificação do registo CNAME
3.3	Adição de nova conta de acesso FTP
4	**Eliminação de *virtual host***
4.2	Eliminação do registo CNAME
4.3	Eliminação das contas de acesso FTP associadas

Identificado como o padrão mais utilizado nas aplicações instaladas, a criação de um *virtual host* e parametrização do mesmo apresentou-se como requisito de implementação obrigatória com garantias de automatização de etapas necessárias como a adição de um registo CNAME ao servidor de DNS caso se trate de um servidor DNS *master* e criação de uma conta de acesso File Transfer Protocol (FTP) (tabela 1.5) para posterior acesso ao diretório do *virtual host* criado.

Tabela 1.6: Tabela de requisitos do serviço de gestão *DNS*

E	**Gestão de registos DNS**
1	**Informação sobre registo DNS**
1.1	Informação sobre registos A
1.2	Informação sobre registos PTR
1.3	Informação sobre registos MX
1.4	Informação sobre registos NS
1.5	Informação sobre registos TXT
1.6	Informação sobre registos SR
2	**Criação de registo DNS**
2.1	Criação de registo A
2.2	Criação de registo PTR
2.3	Criação de registo MX
2.4	Criação de registo NS
2.5	Criação de registo TXT
2.6	Criação de registo SRV
3	**Modificação de registo DNS**
3.1	Modificação de registo A
3.2	Modificação de registo PTR
3.3	Modificação de registo MX
3.4	Modificação de registo NS
3.5	Modificação de registo TXT
3.6	Modificação de registo SRV
4	**Eliminação de registo DNS**
4.1	Eliminação de registo A
4.2	Eliminação de registo PTR

4.3	Eliminação de registo MX
4.4	Eliminação de registo NS
4.5	Eliminação de registo TXT
4.6	Eliminação de registo SRV

Também de grande importância, o serviço de gestão dos registos DNS do servidor procurou-se abranger o máximo de tipos de registos suportados pelo sistema sendo assim garantido um controlo mais alargado de zonas. (tabela 1.6)

Tabela 1.7: Tabela de requisitos do serviço de gestão da lista de *bugfixes*

F	**Gestão de entradas na lista de *bugfixes***
1	**Informação sobre entrada na lista de *bugfixes***
2	**Adição de entrada na lista de *bugfixes***
3	**Modificação de entrada na lista de *bugfixes***
4	**Eliminação de entrada na lista de *bugfixes***

Por fim, o registo de instalação de uma aplicação com informação referente ao pacote instalado e ao processo de instalação apresenta-se também de importância elevada, garantindo assim a manutenção correta e atualizada de uma lista de aplicações instaladas (tabela 1.7).
Implícito na lista de requisitos encontra-se o processo de desinstalação de aplicações que, pela sua simplicidade e necessidade de um menor grau de controlo, poderá ser implementado utilizando as ferramentas providenciadas mas com maior liberdade na criação de estruturas de desinstalação próprias.
Será também importante mencionar o fato de, em ordem a manter a estrutura do sistema IPBrick e o fluxo de dados aconselhável apresentado posteriormente na secção 3.1.3, surgirá a necessidade da criação de bibliotecas específicas ainda não disponíveis. Excetuando o serviço de instalação por ser exclusivo da API a desenvolver e o serviço de gestão de contas de utilizadores por já se encontrar disponível uma biblioteca IPBrick para o efeito, prevê-se que todos os outros mencionados na análise de requisitos elaborada necessitem da estruturação de novas bibliotecas não sujeitas a análise no presente documento.

Capítulo 2

Apresentação e Contextualização do Problema

2.1 Abordagem ao problema

A abordagem a um problema compreende duas fases. A primeira implica a compreensão e análise do problema quanto aos temas, partes e variáveis envolvidas e a segunda a estruturação de um plano de resolução. Neste capítulo é apresentada a primeira fase da abordagem ao problema deste projeto, identificando as suas partes estruturantes e apresentando e analisando todas as opções disponíveis para para uma dessas partes, realizando uma comparação objetiva com base no contexto presente. São então primeiro definidas as noções de aplicação *third party* e API entrando em detalhe na constituição da última e todas as opções disponíveis. Utilizando também o conhecimento sobre a API de comunicação IPBrick existente e um modelo de software escolhido, será então possível posteriormente definir um esquema para o produto final. Por fim, serão identificadas um conjunto de ferramentas e tecnologias a utilizar no seu desenvolvimento assim como a sua importância para o projeto.

2.2 Aplicações *third party*

Designa-se por aplicação *third party* o *software* distribuído por outra entidade que não a da(s) plataforma(s) a que se destina podendo consistir em programas completos ou apenas *plugins* ou extensões que adicionam funcionalidades a outro programa. Sendo o sistema IPBrick um sistema baseado em Debian, a instalação de uma aplicação no mesmo procede-se através do uso de pacotes Debian. De maneira a compreender a estrutura de uma aplicação desenvolvida para IPBrick é preciso então analisar a estrutura de um pacote deste tipo. Esta análise é feita na secção 2.8.2.

2.3 *Application Programming Interface*

Application Programming Interface ou API é o nome atribuído a uma interface que define o modo como outras aplicações podem comunicar com a aplicação com esta interface implementada, reunindo um conjunto de instruções e padrões para o seu acesso.
A criação de uma API por parte de uma aplicação permite assim a integração de recursos e funcionalidades da mesma por uma outra sem necessidade do envolvimento da segunda nos detalhes de funcionamento

interno da primeira, razão pela qual tem sido uma solução adotada por muitas empresas no seu *software* com beneficio para quem vê o seu sistema ser base para outro tipo de soluções e para quem consegue desenvolver os seus produtos de forma mais rápida e eficiente utilizando os recursos disponíveis de outro.
Uma API é, na prática, usualmente criada na forma de uma ou mais bibliotecas com especificações de rotinas, estruturas de dados, classes de objetos e variáveis podendo também ser apresentada, no caso de APIs desenhadas para serviços *web*, como uma definição de acessos e pedidos remotos.
Em programação apresenta-se como uma fronteira entre duas entidades que permite a transferência de informação entre ambas, um "contrato" de parâmetros que ambos devem implementar para que a comunicação entre ambas as partes seja viável. Podem então descrever, sucintamente:

1. As mensagens que são compreendidas pelo objeto;
2. Os argumentos que podem ser enviados nas mensagens;
3. O tipo de resultados que as mensagens retornam.

Posteriormente neste capítulo serão aprofundadas duas partes que compõem uma API e que as distinguem na sua utilidade. Uma análise sobre o formato das mensagens é feita na secção 2.4 e um estudo sobre a arquitetura e soluções existentes é efetuado na secção 2.5.
Dada a implementação de serviços no sistema IPBrick proceder-se por meio de serviços *web* é relevante e adequado projetar a API de acordo com este paradigma. Desta abordagem resulta não só uma ferramenta mais adequada mas também mais generalista à qual se poderá, posteriormente, estender o uso. É portanto importante ter em conta dois pontos importantes [6]

- Neutralidade de aplicação
- Neutralidade de implementação

Segundo a analogia de Nick Gall [6], é conveniente olhar para a arquitetura de uma API como uma ampulheta. Por um lado, é necessário desenvolver uma estrutura neutra ao nível da aplicação, podendo abranger o máximo de entidades possíveis. A este ponto Gall refere-se como o topo da ampulheta pois permite uma maior abrangência, uma maior quantidade de recursos. Por outro, é também necessário ter em conta a neutralidade de implementação, uma abordagem flexível para todos os tipos de tecnologias disponíveis e meios de comunicação, sendo este o fundo da ampulheta de Gall. Porém, embora um protocolo de aplicação mais genérico exerça um efeito maior na rede e deva ser o foco principal, um equilíbrio entre as duas partes trará um maior beneficio do que a implementação baseada em apenas uma.
No que toca à estrutura de uma API, é possível dividi-la em duas partes fundamentais:

- o formato das mensagens na transferência;
- a arquitetura

Tendo como base os pontos referidos anteriormente é então feita uma análise das opções para ambas as partes disponíveis atualmente.

2.4 Formatos de Mensagens

Durante muitos anos, Extensible Markup Language (XML) foi o *standard* e considerado por muitos a solução para o problema de serialização na transferência de dados estruturados entre aplicações, eliminando a necessidade de *parsers* especializados para cada formato de dados criado por uma entidade. No entanto nas últimas décadas novos formatos foram aparecendo como opções válidas e com características diferentes

com a intenção de suplantar o formato XML e algumas das suas falhas, principalmente no que toca à sua eficácia na utilização de recursos.
Na tabela 2.1 é possível analisar o número de interfaces que utilizam ou utilizaram um dado formato de dados, com valores de Janeiro de 2014, segundo a ProgrammableWeb [5].

Tabela 2.1: Número de aplicações por formato de mensagem [5]

Formato	**Número de interfaces**	**Percentagem de interfaces**
XML	6006	48,69
JSON	5117	41,48
RSS	242	1,96
HTML	236	1,91
CSV	227	1,84
PHP	152	1,23
Text	138	1,12
RDF	74	0,60
YAML	63	0,51
KML	57	0,46
Outros	24	0,19
Total	12336	100

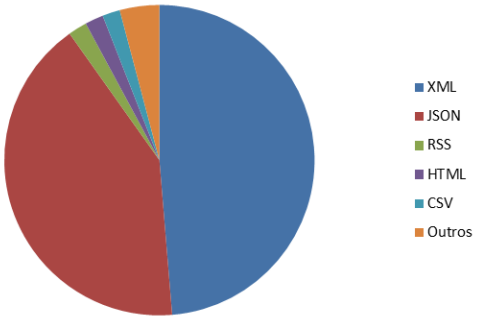

Figura 2.1: Distribuição de formatos de mensagens em aplicações

Como é possível verificar, dois valores destacam-se dos outros, sendo um o previamente referido XML e o outro JavaScript Object Notation (JSON), um formato apresentado 3 anos após o primeiro ter sido considerado um recomendação World Wide Web Consortium (W3C), cuja popularidade tem vindo a crescer. Embora todos os formatos disponíveis sejam válidos de serem utilizados, é necessário primeiro ter em conta a natureza de alguns deles de maneira a tomar uma decisão mais correta para um determinado contexto.

- Resource Description Framework (RDF), Keyhole Markup Language (KML), HyperText Markup Language (HTML) e Rich Site Summary (RSS) são formatos baseados em XML;
- Os formatos Comma-Separated Values (CSV) e *Text*, pela sua simplicidade, não oferecem muitas vantagens de utilização ;
- RSS está desenhado para um tipo de funcionalidade específico: alertas;
- PHP Hypertext Preprocessor (PHP) é versátil mas demasiado generalista e potencialmente complexa.

Tendo em conta as estatísticas, o suporte e ferramentas fornecidas e as observações indicadas, optou-se pela análise mais aprofundada de XML e JSON, colocando de lado as outras opções.

2.4.1 XML

XML (eXtended Markup Language) é uma linguagem de representação universal de dados com sintaxe baseada na utilização de marcadores estruturados em forma de árvore. Esta componente permite não só descrever o conteúdo de um dado objeto ou entidade mas também a sua estrutura, garantindo assim uma fácil e rápida interpretação quer por humanos, quer por máquinas. No âmbito de transferência de informação estruturada é usada maioritariamente na execução de Remote Procedure Call (RPC)s e serialização de objetos.
Utilizado com o intuito de homogeneizar o formato dos dados na transferência dos mesmos entre aplicações foi amplamente adotado como uma das principais soluções neste assunto e base de muitos outros novos formatos.
A capacidade de utilização de esquemas e *namespaces* garante ao XML um carácter flexível, aberto, genérico e versátil, permitindo adaptação aos mais variados contextos. Contudo, a quantidade de funcionalidades disponíveis e a sua versatilidade têm custo de largura de banda, armazenamento e poder de processamento [7]. É na tentativa de resolver estas questões que surge a linguagem JSON.

2.4.2 JSON

JSON (JavaScript Object Notation) define um formato de dados leve para transferência de objetos baseado em pares atributo-valor. Na sua sintaxe constam 4 tipos primitivos (*strings*, inteiros, *booleans* e *null*) e 2 tipos estruturados (objetos e vetores)[8]. Foi desenhado com o intuito de garantir independência da linguagem com a qual se relaciona e permitir uma rápida e eficaz conversão, compreensão e interpretação.
Ao analisar código em JSON é intuitivo associar a sua aparência, estrutura e sintaxe ao dos vetores de muitas linguagens de programação disponíveis, razão pela qual foi facilmente adotada como uma alternativa ao XML por parte de diversos programadores que viram no novo formato um modo mais natural e simples de definir dados.

2.4.3 XML vs JSON

Debates com este tema são recorrente sem existir um consenso das partes integrantes. É, contudo, possível de se afirmar que ambos os formatos são válidos, possuem pontos fortes e pontos fracos e devem ser escolhidos consoante a situação.
Nas tabelas 2.2 e 2.3 são apresentadas algumas vantagens e desvantagens identificadas e analisadas de cada linguagem tendo em conta um comparativo.

Tabela 2.2: Vantagens e Desvantagens de XML

XML	
Vantagens	**Desvantagens**
Fácil leitura	Excesso de carateres nos seus elementos por necessidade de auto-descrição
Maior flexibilidade	Demasiado liberal tornando-se fácil de complicar
Utilização de *namespaces* evita colisões de nomes	Mais lento na transferência (mais símbolos)
Esquemas fornecem validação de tipos de dados e estruturas	Necessidade de manipulação de dados na receção

Tabela 2.3: Vantagens e Desvantagens de JSON

JSON	
Vantagens	**Desvantagens**
Sintaxe mais simples	Menor flexibilidade efeito da simplicidade da sintaxe
Menor *overhead*	Validação de entrada não suportada por defeito
Integração natural com algumas linguagens, nomeadamente Javascript	

Em suma, é correto afirmar que para um maior controlo e validação das mensagens e mais funcionalidades extra XML reúne as melhores características. No entanto, se o foco for na eficácia, simplicidade e rápida integração e aprendizagem a escolha deve recair mais para JSON. É importante salientar que JSON não se apresenta como um formato auto-descritivo ao contrário de XML. Isto leva a uma maior necessidade de incorporação de informação no formato XML o que se traduz em algumas das desvantagens apresentadas no comparativo apresentado. No contexto deste projeto não se apresenta a necessidade de utilização de um formato auto-descritivo, razão pela qual é considerado JSON como uma opção e todas as considerações são tomadas baseadas nesse aspeto. É, por fim, ainda importante referir um aspeto importante: ferramentas disponíveis. Sendo uma solução mais antiga e, durante muitos anos, maioritariamente utilizada em todos os contextos sobretudo o empresarial, XML apresenta um muito maior número de ferramentas e mecanismos de suporte. No entanto, com a popularidade emergente de JSON, também o número de ferramentas e apoio tem vindo a aumentar, sendo em muitos casos mais atualizadas do que as do seu "rival".

2.5 Arquiteturas orientadas a serviços

Service Oriented Architecture (SOA) é o nome designado a um estilo de arquitetura de *software* baseado na apresentação de funcionalidades a outras aplicações por meio de serviços.
Um serviço consiste numa ação executada por um prestador de serviços a pedido de um consumidor. Esta é uma metodologia que permite, do ponto de vista logístico, a separação de responsabilidades em diferentes entidades que comunicam entre si podendo ser programados em linguagens e sobre plataformas distintas.
O principal atributo que define um *software* baseado nesta arquitetura é o pouco ou nenhum conhecimento acerca do funcionamento de cada um dos componentes que providenciam cada serviço, apenas das funcionalidades que oferecem por meio de uma interface pública. É possível comparar um componente com uma caixa-preta, encapsulando o serviço que presta, tornado-o apenas acessível através da interface correspondente [9]. Uma representação visual da estruturação de um serviço deste género pode ser visto na figura 2.2

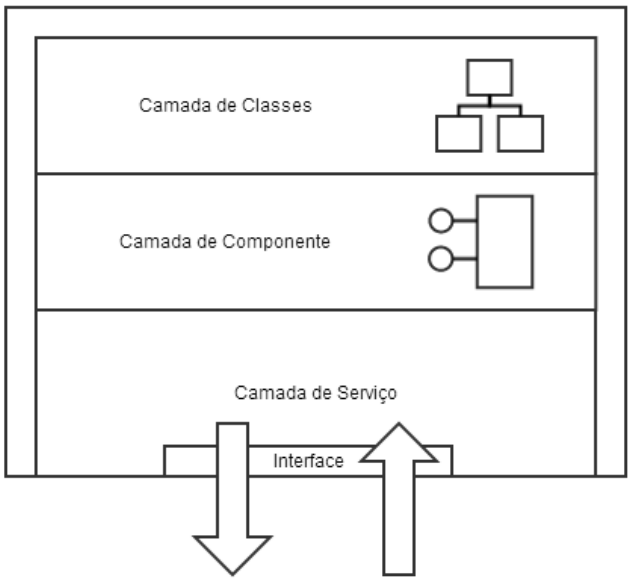

Figura 2.2: Estrutura de uma aplicação SOA [1]

Aplicações desenhadas com este propósito existiram muito antes da arquitetura SOA ter sido definida mas só posteriormente foram concebidos e utilizados conjuntos de tecnologias que se tornaram padrões em aplicações deste género, sendo as mais comuns o protocolo baseado em XML e Hypertext Transfer Protocol (HTTP) designado Simple Object Access Protocol (SOAP), a arquitetura baseada em recursos Representational State Transfer (REST), o esquema de XML XML Schema (XSD) e a norma de descrição de serviços Web Service Description Language (WSDL) entre outras. Estas tecnologias serão analisadas com mais detalhe posteriormente.

As características desta arquitetura levam-na a ser uma solução eficaz no planeamento de aplicações atuais. O mundo em constante mudança que é o mundo dos dias de hoje exige uma constante adaptação a essa nova mudança, modificando os recursos que ontem eram adequados mas que hoje já não o são. A arquitetura SOA garante a flexibilidade necessária para alterações rápidas e eficazes como resposta a fatores internos (reestruturações, aquisições e redução de custos) ou externos (exigência de clientes e competitividade).

A sua estruturação em componentes também permite quer um investimento mais ponderado por serviços baseados em recursos existentes quer um desenvolvimento mais rápido e segmentado apenas desses serviços. O seu foco mais direcionado para a especificação do que para a integração de um componente permite também a re-utilização mais eficaz de serviços com menos duplicação de recursos o que se traduz em menos custos.

É no entanto possível identificar alguns aspetos menos positivos desta arquitetura como é o caso da necessidade de uma redefinição e reestruturação total de uma aplicação que não seja mas que se pretenda ser baseada em SOA o que leva a, dependendo do tamanho, investimentos monetários e de tempo não viáveis. É também opinião de um segmento dos técnicos e teóricos especializados nesta temática que grande parte das tecnologias em que SOA se baseia não sejam de fácil aprendizagem e especialização, sendo ainda normalmente necessárias mais do que uma para implementação. Por fim, alguns estudos mostram indícios que apontam para um menor grau de eficácia desta tecnologias do que outras opções em termos de latência, utilização do Central Processing Unit (CPU) e largura de banda, principalmente as baseadas em XML [10].

Web services são as implementações mais comuns de componentes de aplicações SOA. Como o próprio

nome indica, são serviços fornecidos através da rede, entre aplicações, com base em mensagens de pedido e resposta. Têm como principais atributos a utilização de protocolos *open standard* nomeadamente HTTP e XML, sendo completos, suficientes e passiveis de serem utilizados por outras aplicações [11]. Por serem baseados em SOA, têm de ser desenhados com vista a garantir independência da plataforma e da linguagem de desenvolvimento, permitindo interoperabilidade entre soluções de diferentes tipos.

Tabela 2.4: Número de aplicações por estilo [5]

Estilo	**Número de interfaces**	**Percentagem de interfaces**
REST	6888	68,88
SOAP	2130	21,30
Javascript	585	5,85
XML-RPC	200	2,00
HTTP GET/POST	96	0,96
AtomPub	29	0,29
JSON-RPC	28	0,28
GData	20	0,20
RSS	14	0,14
XMPP	8	0,08
OSCAR	2	0,02
Total	10000	100

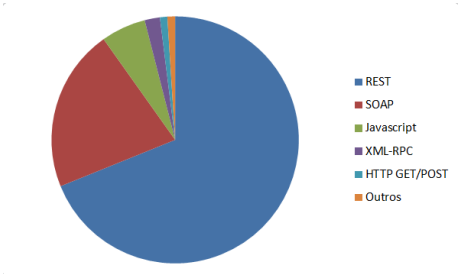

Figura 2.3: Distribuição de estilos de aplicações

Atualmente, como se pode verificar na tabela 2.4 que lista os estilos utilizados em diferentes *web services* segundo a ProgrammableWeb [5], os estilos mais utilizados de *web services* baseiam-se ou no protocolo SOAP ou na arquitetura REST.

2.5.1 SOAP

SOAP (Simple Object Access Protocol) é o protocolo responsável por codificar as mensagens XML trocadas entre as aplicações de maneira a serem compreendidas pelas duas partes, independentemente das especificações de cada uma delas e da rede que as liga, tendo sido a Microsoft e a IBM as primeiras empresas a apostar no seu desenvolvimento com contribuições posteriores de outras companhias como a Ariba, Compaq, HP e Lotus [12] [13]. Este protocolo introduz a necessidade das mensagens serem transportadas num envelope SOAP e permitem a utilização do serviço de descrição de interfaces WSDL [14].
Um envelope SOAP é a camada XML raiz que encapsula a mensagem, contendo os campos de cabeçalho (opcional) e corpo da mensagem (obrigatório). Cada um destes campos contém sub-elementos nos quais a informação está estruturada. O cabeçalho poderá conter informação relacionada com a aplicação,

direcionada a nós intermediários enquanto que o corpo da mensagem contém os dados da transmissão ponto-a-ponto.

Pedido SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
                <m:GetStockPrice>
                        <m:StockName>IBM</m:StockName>
                </m:GetStockPrice>
        </soap:Body>
</soap:Envelope>
```

Resposta SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
                <m:GetStockPriceResponse>
                        <m:Price>34.5</m:Price>
                </m:GetStockPriceResponse>
        </soap:Body>
</soap:Envelope>
```

O formato das mensagens SOAP assume uma estrutura em XML, podendo mesmo dizer-se que as primeiras são especificações da segunda. Isto é necessário pois, assegurando que cada entidade participante consegue interpretar e gerar mensagens SOAP é possível assegurar a comunicação e transferência de informação estruturada entre elas.

A norma WSDL define um formato de esquema XML que permite a descrição de um serviço (as suas funções, parâmetros e retornos) de maneira a que um cliente possa obter todas as informações acerca do mesmo, gerar e interpretar as mensagens de acordo com os requisitos pedidos [15]. WSDL é versátil o suficiente para permitir a descrição de entidades e as suas mensagens independentemente dos formatos de mensagem ou protocolos de rede usados para comunicar [16]. É garantido assim não só o conhecimento das

Types	definição dos tipos de dados com base em esquemas
Message	definição dos dados a serem transmitidos
Operation	definição das funcionalidades do serviço
Port Type	identificação do conjunto de operações integradas nas entidades
Binding	identificação do protocolo e formato de dados utilizado por um Port Type
Port	definição de uma entidade através do *binding* e do endereço de rede
Service	conjunto de entidades relacionadas

Tabela 2.5: Elementos de um ficheiro WSDL

funcionalidades do serviço mas como a especificação correta e completa dos pedidos. Um ficheiro WSDL é composto por 7 elementos necessários para definir um serviço, apresentados na tabela 2.5 [16].

No anexo B é possível visualizar um exemplo de um ficheiro WSDL.

Existem no total 5 padrões possíveis de mensagens SOAP derivados de pares de estilos e métodos de serialização de dados. Uma mensagem pode ser formatada segundo o estilo RPC ou *documento*. O primeiro começou por ser o mais adotado pela sua simplicidade de compreensão, lógica e facilidade de mapeamento para as tecnologias de computação tradicionais. No entanto a liberdade de estruturação do corpo da mensagem do estilo *documento* em XML nativo e o fraco desempenho e escalabilidade do estilo RPC convenceram a maioria dos utilizadores a mudar a sua escolha. Atualmente, na versão 1.2, e ao contrário de versões anteriores, a utilização do estilo RPC é opcional e o padrão é *documento*.

É também prática comum associar estes dois estilos com dois métodos de serialização de dados: *literal* e *codificado*. O primeiro baseia-se na utilização de esquemas XML padrão para definir a estruturação do conteúdo enquanto que o segundo utiliza regras SOAP específicas para esta questão. O fato de o método *codificado* não fazer parte dos padrões Web Services Interoperability (WS-I) (organização responsável por estabelecer boas práticas para interoperabilidade entre *web services*) levou a uma maior adesão da opção alternativa.

No entanto, uma das críticas apontadas ao padrão *documento/literal* residia na dificuldade de leitura e compreensão de uma mensagem por falta de identificação clara do método e parâmetros utilizados. Em ordem a solucionar este problema, foi criada uma extensão deste padrão ao qual se deu o nome de *documento/literal encapsulado* que adicionava elementos para auxiliar a leitura da mensagem. No entanto, esta adição traduz-se numa maior complexidade de código necessário para a implementar.

2.5.2 REST

REST (Representational State Transfer) foi o nome designado por Roy Fielding [17], um dos principais autores da especificação do protocolo HTTP, em 2000 para a arquitetura de *web services* baseado no mesmo protocolo e assente num conjunto de conceitos definidos. Esta arquitetura restringe as ações de uma interface a um conjunto de verbos do protocolo HTTP (*GET, POST, PUT e DELETE*) com a justificação de que estas são as ferramentas suficientes para a construção de um *web service* cujo acesso se tornou generalizado e que remove a necessidade de implementação de um protocolo extra por cima da base de comunicação HTTP tornando a sua utilização mais leve e rápida.

A proliferação e abundância de utilizadores consumidores de *web services* criou a necessidade de simplificação dos protocolos utilizados. REST associado com JSON surgiu como a solução mais adequada para o problema. Com requisitos mais baixos de largura de banda e utilização do processador, grandes empresas prestadoras de serviços através da Internet como a Google, Amazon, Facebook e Twitter, cujo valor de

Figura 2.4: Estrutura de uma comunicação na arquitetura REST [2]

tráfego era elevado, adotaram esta arquitetura nos seus sistemas com vista a aumentar a eficácia dos mesmos. Outro acontecimento que despoletou a procura por serviços baseados em REST foi o aparecimento e proliferação de utilizadores de dispositivos móveis. A limitação na largura de banda e capacidade de processamento do ponto de acesso reforçou a necessidade de desenvolver *web services* com menos exigências. Uma arquitetura que trata cada pedido de maneira independente como o REST permite também distribuir a carga por servidores de maneira a melhorar o desempenho, reduzindo e, em alguns casos, eliminando a coordenação entre servidores [17]. Alguns casos de utilização de serviços baseados nos princípios REST incluem o serviço Google Maps, o motor de pesquisa Google e a API do serviço Twitter.
Os pontos caracterizantes desta arquitetura identificadas pelo seu autor são:

1. Estrutura Cliente-Servidor
2. Processamento independente de pedidos
3. *Caching*
4. Sistema estruturado por camadas
5. *Code-on-demand*
6. Interface Uniforme

Roy Fielding começou por definir a primeira restrição a REST como a necessidade de uma estrutura cliente-servidor[17]. Esta estrutura traz os benefícios da separação entre lógica e implementação, aumentando a portabilidade da aplicação e a escalabilidade da mesma, permitindo que os seus componentes possam evoluir de maneira independente.
A segunda característica, como referido anteriormente, além de garantir uma melhor escalabilidade por simplificar a gestão de recursos no decorrer de pedidos [2], também torna o serviço mais fiável e aumenta a capacidade de um terceiro componente poder gerir uma comunicação. No entanto, a existência de informação redundante em cada pedido afeta a eficácia da rede, sendo essa uma das consequências negativas cuja característica 3 procurou atenuar. A identificação da possibilidade do cliente manter um pedido em memória permite a reutilização de informação desse no processamento de pedidos seguintes.
A característica 5 refere-se à necessidade de encapsulamento serviços em camadas, cada camada apenas tendo alcance das camadas adjacentes. Uma implementação deste género promove a simplicidade da aplicação e a independência dos seus componentes, colocando um limite na sua complexidade. No entanto

estes sistemas aumentam a latência das comunicações por introdução de fronteiras de processamento algo que pode ser atenuado com utilização de memórias de *caching* em intermediários.

A utilização de *code-on-demand* é uma característica opcional de REST na qual um componente do cliente tem acesso a um conjunto de recursos mas não ao método de processamento ideal. Neste caso, o cliente executa um pedido a um servidor pelo código que permite manipular os recursos e executa-os localmente. Esta característica aumenta a simplicidade de implementação de um serviço e de adição de novas funcionalidade ao mesmo.

Por fim, interfaces uniformes permitem a simplificação da arquitetura e separação dos serviços da implementação dos mesmos. Em REST, esta característica é conseguida por implementação de 4 restrições às interfaces:

1. Identificação de recursos
2. Manipulação de recursos através de representações
3. Mensagens que se auto-descrevem
4. Mudança de estado da aplicação através de hiperligações

1. Identificação de recursos Um recurso é um entidade (física ou conceptual) providenciada por uma aplicação com um identificador Uniform Resource Identifier (URI). Na arquitetura REST existe a necessidade de definir regras para um identificador sendo elas:

- A semântica do mapeamento de um URI para um recurso não pode mudar
- A identidade de um recurso é independente do seu valor
- O fornecedor de um recurso apenas é responsável por manter a validade da semântica de um URI
- Um URI não deve conter nenhuma referência ao tipo de recurso

2. Manipulação de recursos através de representações

Uma representação de um recurso contém dados indicadores do estado desse mesmo recurso, podendo este ser representado de várias maneiras, consoante o pedido efetuado pelo cliente. Em REST existem dois tipos de estados: o estado do recurso e o estado da aplicação sendo que a última indica o caminho percorrido pelo cliente através da aplicação. O primeiro é mantido no lado do servidor enquanto que o segundo é guardado do lado do cliente.

3. Mensagens auto-descritivas

Esta condição obriga a que todos os detalhes para compreensão e processamento de uma mensagem sejam incluído na própria, garantindo uma comunicação com processamento independente de mensagens.

4. Mudança de estado da aplicação através de hiperligações

Teoricamente, uma arquitetura REST providenciaria aos seus clientes juntamente com a resposta pedida, todos os URI possíveis de serem acedidos e relevantes para o pedido efetuado. A esta ideologia foi dado o nome de *Hypermedia as the Engine of Application State* ou simplesmente HATEOAS. Desta maneira o servidor iria apresentando caminhos que o cliente iria escolhendo à medida que ia efetuando os pedidos. No entanto, esta é possivelmente a restrição mais polémica das quatro, sendo muitas vezes considerada uma ideologia da arquitetura e boa prática mas nem sempre adequada e possível de ser implementada em todos os contextos.

2.5.3 SOAP vs REST

Desde a introdução da arquitetura REST que o debate se mantém: seria esta abordagem à implementação de *web services* melhor que o já estabelecido e fiável SOAP? Muitos defendem que, sendo estritamente baseado em tecnologias básicas da Internet como o protocolo HTTP e a definição de recursos por URIs, a implementação REST é a mais adequada para um serviço que se diz ser para utilização através da rede. Atualmente, apesar do crescimento da utilização da nova arquitetura por parte de empresas em todo mundo e da criação ou adaptação de ferramentas para o seu suporte, é possível assumir que cada implementação tem o seu espaço e que apenas dependendo da situação é possível identificar que uma é melhor do que a outra. Nesta secção será apresentada uma comparação do protocolo SOAP e da arquitetura REST nas temáticas de objetivo, protocolos utilizados, escalabilidade, flexibilidade, ferramentas disponíveis e segurança.

2.5.3.1 Objetivo

A principal diferença entre REST e SOAP é, acima de tudo, o objetivo de cada uma delas. SOAP desde sempre foi vocacionado para o acesso a operações de um sistema enquanto que REST apresenta-se direcionado para o acesso aos seus recursos. Na utilização de operações de um sistema através do protocolo SOAP, é necessário que ambas as partes tenham conhecimento do método de utilização das mesmas, daí a necessidade de descriminação de parâmetros e valores de retorno por meio de um documento de descrição da interface (documento WSDL apresentado em 2.5.1) criado com base em normas estipuladas de maneira a poder ser compreendido pelo cliente que necessitar de utilizar o serviço. A manipulação de recursos de um serviço através da arquitetura REST é restrita à utilização dos verbos HTTP pois estes podem ser relacionados com as operações Create Read Update Delete (CRUD), operações básicas usualmente utilizadas em bases de dados relacionais. Um pedido a um serviço baseado em REST é então definido pelo URI do recurso a utilizar e pelo tipo de operação que se quer efetuar sobre esse recurso. Esta restrição pode parecer limitadora mas é possível, na maior parte dos casos, definir um conjunto de recursos de um serviço e garantir todas as funcionalidades do mesmo através dos verbos fornecidos pelo protocolo HTTP. Outro ponto em que ambas as tecnologias assumem orientações distintas é no modelo de comunicação. Enquanto que a arquitetura REST assume um modelo de comunicação ponto-a-ponto, SOAP aproxima-se mais a um modelo de comunicação *end-to-end*, mais adequado a ambientes de sistemas distribuídos onde as mensagens podem ser reencaminhadas através de pontos intermediários. O modelo de comunicação de cada uma das tecnologias reflete-se noutras das suas características, nomeadamente nos modelos de segurança, apresentados com mais detalhe na secção 2.5.3.6.

2.5.3.2 Protocolos

No caso da tecnologia REST, o protocolo de transporte é restringido ao HTTP, protocolo sobre qual toda a arquitetura assenta. Já no caso da solução SOAP, a gama de protocolos de transporte é bastante vasta, desde o suporte ao mais comum HTTP até protocolos de serviço de mensagens como Simple Mail Transport Protocol (SMTP) e Post Office Protocol (POP), protocolos para sistemas distribuídos como General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP) e Distributed Component Object Model (DCOM) e os conhecidos protocolos de transporte Internet Protocol (IP) Transmission Control Protocol (TCP) e User Datagram Protocol (UDP). A capacidade de manter o protocolo SOAP independente do protocolo de transporte sobre o qual está a funcionar advém da norma *WS-Addressing*, norma esta que define elementos nos cabeçalhos das mensagens com informações de endereçamento para que nós intermediários capazes de

interpretar mensagens SOAP possam processar essas informações e encaminhar as mensagens independentemente do protocolo de transporte utilizado.

2.5.3.3 Escalabilidade

A capacidade de efetuar *caching* de leituras (operação que compreende a maioria de pedidos de clientes na maioria dos serviços disponibilizados na rede) e a performance da solução fazem com que REST seja, de entre as duas opções, a com melhor índice de escalabilidade como é possível verificar no resultado das experiências realizada por G. Mulligan e D. Gračanin em [3]. Na primeira experiência, ao aumentarem o número de pedidos síncronos a uma interface (uma SOAP e outra REST) de um servidor, verificaram que, além de o valor da latência da segunda ser constantemente mais baixa que a primeira, o seu crescimento era linear e não exponencial como o caso do SOAP, como visível na figura 2.5.

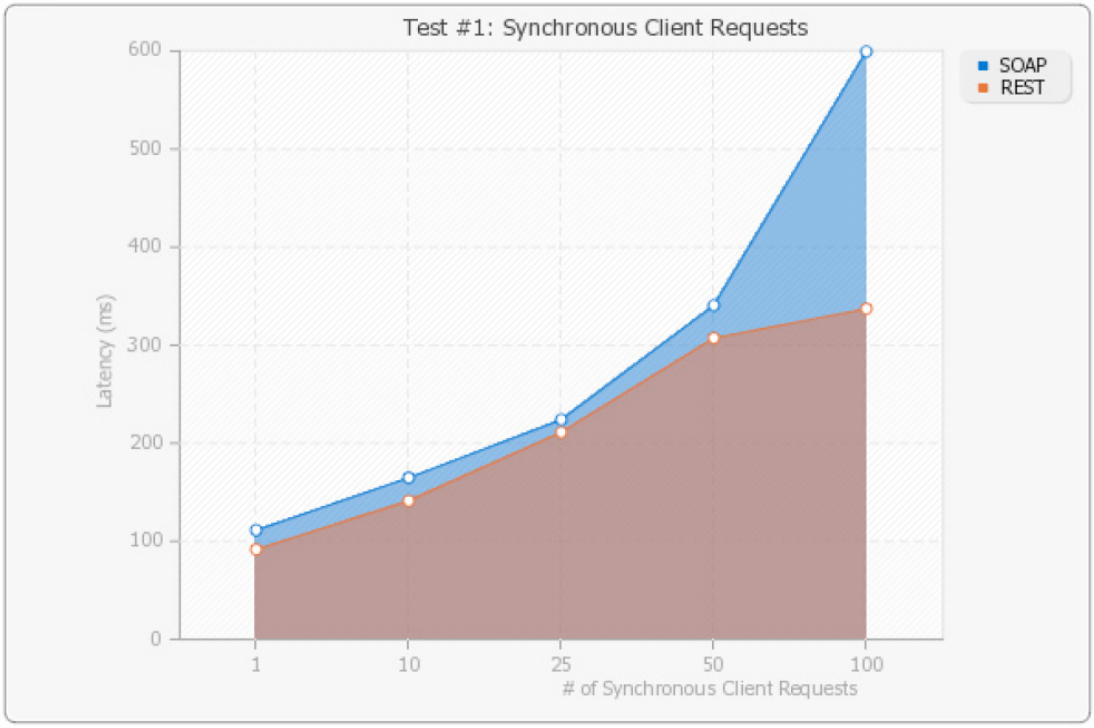

Figura 2.5: Variação da latência por quantidade de pedido síncronos em SOAP e REST [3]

Na segunda experiência, voltaram a comparar a latência obtida com cada uma das soluções bem como o tamanho dos pacotes trocados ao aumentar a complexidade de uma aplicação. É possível identificar na figura 2.6 que, embora os valores de latência para uma aplicação simples sejam mais baixos numa implementação SOAP, rapidamente essa situação não apenas se inverte como se distancia com o aumento da complexidade da aplicação.

No que toca ao tamanho dos pacotes, os valores de ambas as implementações mantêm-se constantemente próximas o suficiente para a comparação ser considerada irrelevante, como visível na figura 2.7. No entanto o facto de, nos testes realizados neste artigo, o corpo das mensagens da arquitetura REST ser em formato XML e não um formato mais leve como JSON mostra que mesmo o caso menos eficaz apresenta melhores resultados do que uma implementação SOAP.

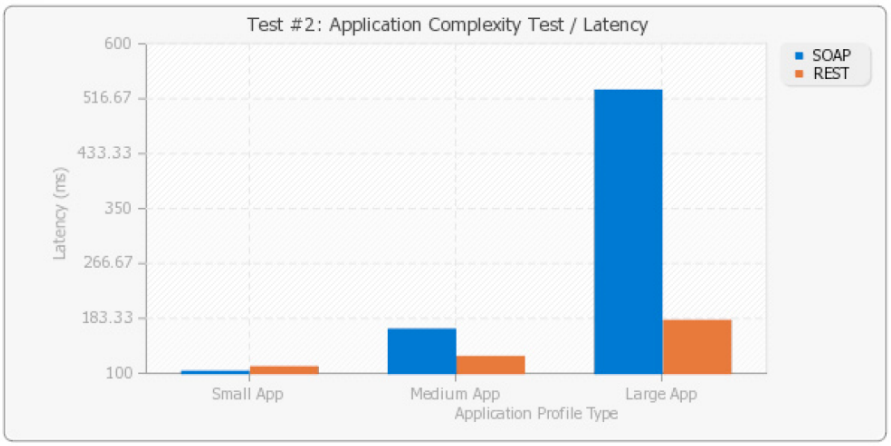

Figura 2.6: Variação da latência por grau de complexidade da aplicação em SOAP e REST [3]

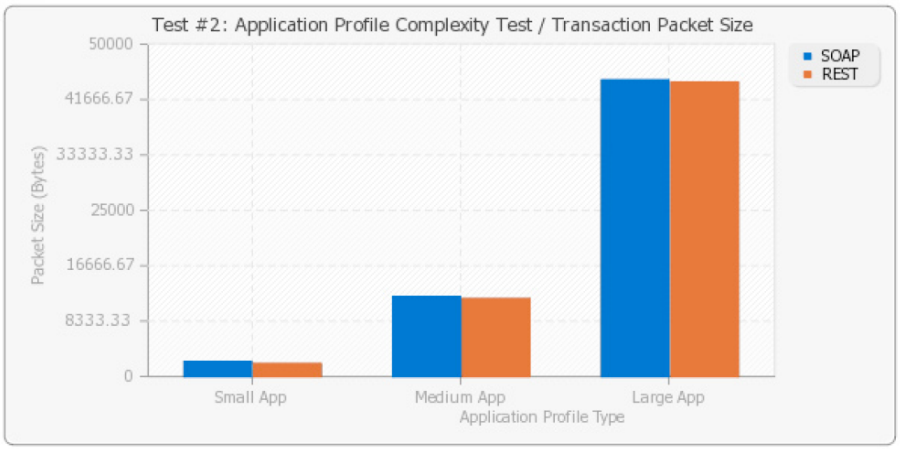

Figura 2.7: Variação do tamanho dos pacote transmitidos por grau de complexidade da aplicação em SOAP e REST [3]

2.5.3.4 Flexibilidade

A principal característica do protocolo SOAP é, atualmente, o ponto que mais é apontado como a sua fraqueza em comparação com a arquitetura REST: o formato e estrutura das suas mensagens. Sendo que garante a compreensão e interpretação correta pelas duas parte da informação transmitida, o aparecimento de uma alternativa que permite a escolha de um outro formato para as suas mensagens colocou a REST numa posição bastante favorável, principalmente desde a crescente utilização de APIs públicas de acesso por navegadores e o aparecimento de formatos mais leves e adequados à interpretação por estes meios de acesso. Por sua vez, a capacidade do protocolo SOAP ser implementado independentemente do protocolo de transporte (como enunciado em 2.5.3.2) é um ponto extra em relação à implementação limitada ao protocolo HTTP de um serviço REST. Existem também diferenças na implementação de cada uma das soluções num servidor. Enquanto que um servidor com API SOAP é mais simples de aplicar pois possui mais ferramentas que aceleram o processo mas tem uma curva de aprendizagem mais acentuada, uma API REST exige mais tempo para implementar mas é mais simples de perceber conceptualmente e de se adaptar ao serviço pois traduz-se quase linearmente aos processos de acesso a bases de dados.

2.5.3.5 Ferramentas

Sendo mais antigo, o protocolo SOAP conta já com uma vasta gama de ferramentas quer para implementação de uma API, para geração de ficheiros WSDL ou para desenvolvimento de um cliente SOAP a partir de um ficheiro WSDL. No entanto, estas ferramentas são necessárias pois o protocolo está bem definido e tem normas que têm de ser seguidas. No que toca a REST, existe apenas a necessidade de manipulação do verbo e do URI de pedidos e respostas HTTP, funções essas incluídas em bibliotecas já nativas na maior parte das linguagens de programação atuais. Mesmo para a interpretação da maior parte dos formatos de mensagens já existem bibliotecas disponíveis, como é o caso da biblioteca SimpleXML para PHP. O único ponto em que existe uma falta de ferramentas para desenvolvimento encontra-se na descrição de uma interface REST o que, embora contradiga a definição da arquitetura segundo o seu pioneiro, é ainda requisitada em algumas situações. Esta situação decorre não só do não apoio da necessidade da existência deste tipo de descrição de interfaces por força do paradigma Hypermedia As The Engine Of Application State (HATEOAS) apresentado como característica fundamental de uma API REST, como também do não estabelecimento de uma norma de descrição deste tipo de interfaces como o caso do WSDL em serviços SOAP.

2.5.3.6 Segurança

Assim como o modelo de comunicação de REST e SOAP são distintos, também a sua abordagem à segurança dessa comunicação. A tecnologia REST herda, por norma, o mecanismo de segurança do seu protocolo de transporte, o HTTP com protocolo de segurança Transport Layer Security (TLS)/Secure Sockets Layer (SSL) ou HyperText Transfer Protocol Secure (HTTPS). Esta implementação do protocolo HTTP garante que as mensagens transmitidas através do canal de comunicação não sejam visualizadas e compreendidas por terceiros. Esta abordagem é, na maioria das vezes, suficiente mas não compreende uma defesa contra ataques às extremidades do canal de comunicação. Para autenticação, foram elaboradas algumas soluções:

1) Autenticação básica HTTP Este é o método mais simples de autenticação no qual são enviadas as credenciais do utilizador, codificadas em Base 64, através do cabeçalho de um pedido HTTP, não sendo necessário qualquer tipo de negociação e acordo prévio [18]. Esta operação não garante confidencialidade das informações trocadas, dependendo por isso de cifragem ao nível da camada de transporte.

2) HTTP Digest Este método veio retirar a necessidade da dependência da camada de transporte para confidencialidade por aplicação de *hashing* nas informações enviadas para autenticação e veio adicionar uma maior resiliência contra Choosen-Plaintext Attack (CPA)s com a inclusão de valores de sessão gerados aleatoriamente (previamente apenas emitidos pelo servidor, mais tarde também emitidos pelo cliente) [18], valores esses que, relacionados com hora de emissão, podem também proteger o sistema contra ataques de personificação. No entanto este método ainda é vulnerável a ataques do tipo *man-in-the-middle* e duplicam o número de mensagens trocadas de duas para quatro.

3) OAuth 2.0 OAuth é uma norma que permite o acesso de uma terceira entidade a recursos protegidos sem troca de credenciais entre esta e o dono dos recursos. Em suma, a entidade que quer aceder aos recursos pede um *request token* que, depois de autenticado pelo dono dos recursos ou por um servidor de autenticação, pode ser trocado por um *access token* que a terceira entidade utilizará para aceder aos recursos pretendidos [19].

4) Autenticação *cookie-based* Através deste método, após autenticação do cliente, uma aplicação guarda um elemento identificador de um utilizador (*cookie*) enviado pelo servidor na sua primeira resposta. A *cookie* é então enviada automaticamente juntamente com cada pedido de um cliente para autenticação automática do mesmo. Por gerar problemas de personificação caso um atacante consiga obter esse elemento, é aconselhável também a utilização de segurança ao nível da camada de transporte, nomeadamente SSL. Aquando da criação da *cookie* do lado do servidor é possível definir parâmetros como intervalo de validade da mesma ou obrigatoriedade de utilização de SSL de maneira a aumentar a segurança do serviço.

5) Certificados de Cliente Da mesma maneira que certificados de servidor servem para autenticação de um servidor a um cliente, também é possível a utilização de certificados para autenticação de um cliente a um servidor. No caso de um cliente possuir um par de chaves (pública e privada), um servidor poderá assinar a chave pública do cliente usado a sua chave privada, passando a identificá-lo com segurança em comunicações posteriores.

6) HMAC Hash-based Message Authentication Code (HMAC) é um mecanismo para autenticação e verificação de integridade de mensagens através da utilização de uma chave secreta partilhada entre cliente e servidor [20]. Ao enviar um valor HMAC do pedido gerado através de uma chave partilhada é possível não só garantir que foi o pedido provém do segundo dono da chave, como é possível confirmar a não alteração do pedido pois o valor é gerado com base no mesmo. Esta abordagem incorre no problema da encriptação simétrica, a de partilha segura de uma chave secreta previamente.

Os serviços SOAP, além de poderem utilizar os mecanismos de segurança providenciados pelo protocolo SSL numa comunicação através de HTTP, podem adicionalmente usufruir dos benefícios da norma WS-Security. Esta norma garante a proteção e mecanismos de autenticação adequados ao contexto de comunicação SOAP, numa extensão *end-to-end*, oferecendo mais opções para verificação e manutenção da integridade e confidencialidade da informação transmitida, como a utilização de *tokens* de sistemas de autenticação Kerberos ou *tokens* Security Assertion Markup Language (SAML). A norma WS-Security também oferece a possibilidade de garantir não-repúdio, algo não possível numa implementação puramente baseada em HTTP. Outras normas WS-* relacionadas e também disponíveis incluem

- WS-SecureConversation para partilha de contextos de segurança, reduzindo assim o excesso de informação entre mensagens;
- WS-Policy permitindo o anúncio de características oferecidas por serviços e definição de requisitos de características por parte de clientes em XML;
- WS-Trust para gestão mais eficaz de *tokens* de segurança;

2.6 API IPBrick

A API disponibilizada atualmente pela IPBrick para aplicações *third-party* está desenhada para funcionar com base no protocolo SOAP e garante funcionalidades de comunicação necessária para o funcionamento das aplicações, como é o caso da obtenção de utilizadores VoIP, envio de fax, obtenção de impressoras entre outros.
Aquando da versão 5.3, o sistema garantia a transferência de uma única *string* entre cliente e servidor, sendo que esse valor era então interpretado localmente e dividido nos diferentes parâmetros dos métodos utilizados. Assim a descrição de cada funcionalidade do serviço era apresentada com um único parâmetro

de entrada do tipo *string* no ficheiro WSDL. Ora esta abordagem não identificava os valores pretendidos e a ordem dos mesmos (havia necessidade de consulta constante do manual de *web service* fornecido) razão pela qual acabou por ser abandonada a favor de uma descrição mais detalhada dos parâmetros esperados, obrigando a uma reestruturação total do ficheiro WSDL do sistema.

Ficou à consideração do aluno a integração da API a elaborar na já existente ou a divisão em duas APIs distintas (uma de integração e outra de comunicação) sendo que, não havendo motivos para haver uma separação das duas, foi escolhida a primeira abordagem.

O fato de toda a lógica da API já existente e uma grande parte da lógica do sistema ter sido desenvolvida na linguagem PHP contribuiu para a escolha da mesma linguagem para elaboração deste projeto de modo a garantir uniformidade e homogeneidade da plataforma.

2.7 Modelo de *software* MVC

Model-view-controller, mais popularmente conhecido pelo acrónimo MVC, surge como um padrão de estrutura de aplicações baseadas em pedidos e respostas em três tipos de elementos com papeis e nível de conhecimento do ambiente diferentes. Foi apresentada pela primeira vez em 1979 pelo professor Trygve Reenskaug para a linguagem de programação Smalltalk sendo só mais tarde, em 1988, sido adotado como um conceito genérico de programação. Os três tipos de elementos diferentes deste modelo são, desde logo, os especificados no seu nome: elementos *Model*, elementos *View* e elementos *Controller*. Os elementos do tipo *Controller* têm como responsabilidade obter os pedidos do cliente, analisar a forma do seu conteúdo e organiza-la de maneira a poder ser enviada para a lógica da aplicação. Este encontra-se reunida em um ou mais elementos *Model*, elementos normalmente algo independentes do ambiente em que a aplicação é inserida e da forma como os dados são fornecidos à aplicação, tendo só o objetivo de tratá-los, aplicá-los e passar os resultados a uma outra entidade, uma entidade *View*. É na entidade *View* que os dados são organizados num formato de resposta previamente negociada com o cliente de maneira a serem compreendidos e, posteriormente, utilizados. A figura 2.8 apresenta um diagrama representativo desta estrutura.

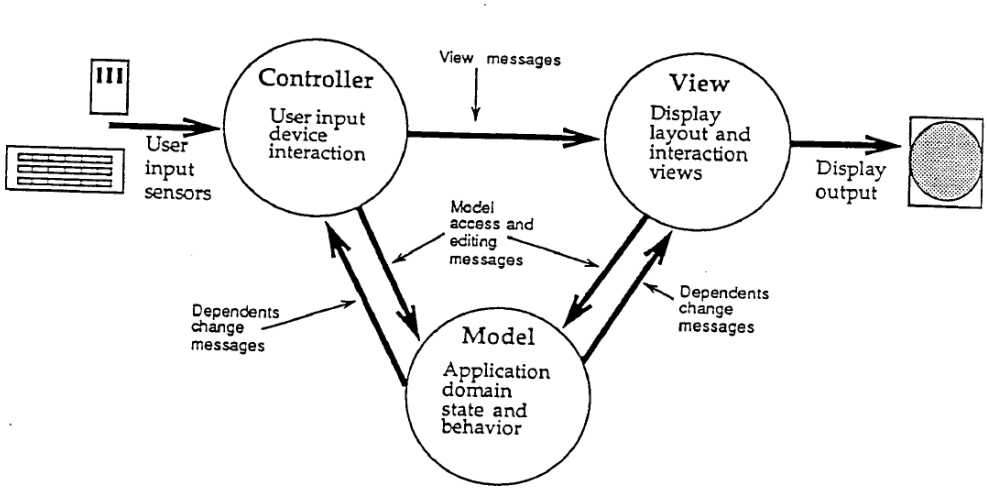

Figura 2.8: Estrutura lógica de uma aplicação baseada em MVC [4]

Esta metodologia garante a divisão de tarefas em blocos específicos o que se traduz no aumento do controlo sobre o fluxo de informação, da escalabilidade da aplicação e da eficácia da mesma derivado da capacidade de controlo de recursos. Além disso, a separação da lógica de uma aplicação dos módulos que providenciam uma interface garante a não necessidade de modificação do seu funcionamento base aquando a integração num novo ambiente, sendo a adaptação feita ao nível dos *Controllers* e dos *Views*. Por fim, é importante evidenciar que a organização mais estruturada do código permite uma melhor manutenção e execução de testes por redução da sua complexidade. Com as evidentes vantagens deste modelo não foi surpresa a sua adoção por populares *frameworks*, nomeadamente Struts e Ruby on Rails, o que se traduziu num aumento considerável da sua divulgação. Atualmente é a estrutura predefinida de grande parte das *frameworks* mais utilizadas como é o caso de Symfony, CakePHP, Zend e CodeIgniter.

2.8 Tecnologias e Ferramentas

2.8.1 SOAP UI

SOAP UI é uma ferramenta *open-source* da SmartBear que permite executar uma vasta gama de testes funcionais a *web services* abrangendo as tecnologias SOAP, REST, WSDL, HTTP, Java Message Service (JMS) e Action Message Format (AMF) com uma interface gráfica simples e intuitiva. Permitindo agrupar conjuntos de pedidos ordenados em casos de teste é possível, graças a esta ferramenta, simular as ações ordenadas realizadas por um utilizador ou aplicação podendo ser ainda definidas regras de validade das respostas de maneira a que, automaticamente, um teste possa ser rejeitado por um determinado passo falhar. O SOAP UI possui também um número considerável de testes de segurança e permite a execução de testes de sobrecarga do serviço, permitindo configurar a quantidade de utilizadores ativos ao mesmo tempo e intervalo entre pedidos entre outros parâmetros. Por fim, apesar de se apresentar como um programa que ajuda no teste e análise de um *web service* de maneira a facilitar a tarefa, também oferece a possibilidade de costumização dos testes por via de *scripting* baseado na linguagem Groove garantindo uma total flexibilidade e alcance para utilizadores mais avançados.

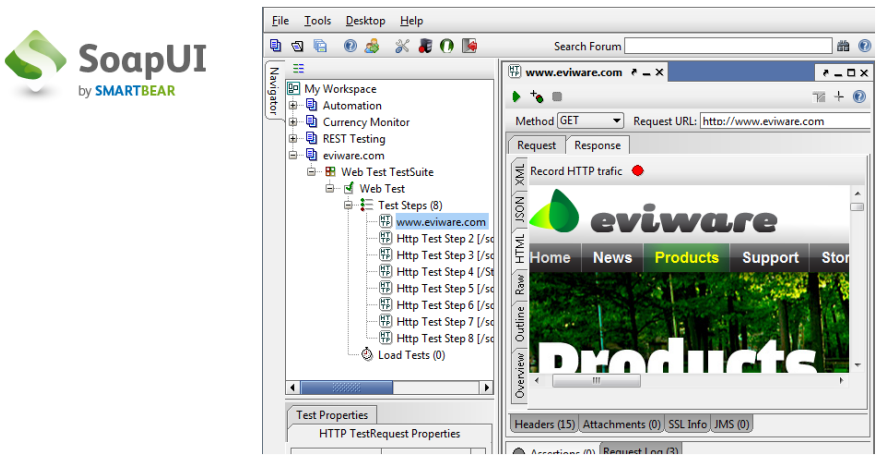

Figura 2.9: Software SOAP UI

debian-binary		identificador da versão do formato .deb do pacote
control.tar.gz		arquivo comprimido com ficheiros de controlo do pacote
	control	contém a informação essencial de controlo
	Source	identificador da origem do pacote
	Maintainer	nome e endereço de e-mail do responsável pelo pacote
	Uploaders	nome e endereço de e-mail de corresponsáveis pelo pacote
	Section	categoria de aplicação do pacote
	Priority	representação da importância para o utilizador da instalação deste pacote
	Build-Depends et al	indicação da dependência de outros pacotes
	Standards-Version	versão mais recente do manual de normas do pacote
	md5sum	*hash* md5 para verificação da integridade do ficheiro
	preinst	*script* para execução antes da extração do pacote
	postinst	*script* para execução após a extração do pacote
	prerm	script para execução antes da remoção dos ficheiros associados com o pacote
	postrm	script para execução após a remoção dos ficheiros associados com o pacote
data.tar.gz		ficheiros da aplicação

Tabela 2.6: Estrutura de um pacote Debian

2.8.2 Pacotes Debian

É por meio de pacotes debian que é possível a instalação de software num sistema Unix, sendo que cada pacote compreende 3 partes essenciais representadas na tabela 2.6. Com vista a examinar um pacote debian, foi feita a transferência de um ficheiro .deb do diretório da Apache [1] e utilizadas as ferramentas disponíveis nos sistemas Unix. A figura 2.10 apresenta a estrutura de mais alto nível do pacote em que é possível identificar os ficheiros debian-binary, control.tar.gz e data.tar.gz. Na figura 2.11 estão apresentados os ficheiros extraídos dos ficheiros .tar.gz apresentados na figura 2.10, onde já se pode identificar o ficheiro control, preinst e postinst. Por fim, a figura 2.12 apresenta o conteúdo do ficheiro control, com as informações relativas ao pacote em questão.

```
[redacted]$ ar tv 'apacheds-2.0.0-M15-amd64
.deb'
rw-r--r-- 0/0       4 Aug 14 20:18 2013 debian-binary
rw-r--r-- 0/0    1538 Aug 14 20:18 2013 control.tar.gz
rw-r--r-- 0/0 9837570 Aug 14 20:18 2013 data.tar.gz
```

Figura 2.10: Estrutura de um pacote debian

A convenção de atribuição de nome a um pacote segue a seguinte forma: <Nome do pacote>_<Versão>-<Versão de Revisão>_<Arquitetura Debian>.deb

[1] https://directory.apache.org/apacheds/download/download-linux-deb.html

```
[redacted]$ ls -la
total 40
drwxr-xr-x  5 joao joao  4096 Ago 14  2013 .
drwxr-xr-x 59 joao joao 12288 Fev 15 18:45 ..
-rw-r--r--  1 joao joao   553 Ago 14  2013 control
drwxr-xr-x  3 joao joao  4096 Ago 14  2013 etc
drwxr-xr-x  3 joao joao  4096 Ago 14  2013 opt
-rwxr-xr-x  1 joao joao  1467 Ago 14  2013 postinst
-rwxr-xr-x  1 joao joao  1532 Ago 14  2013 prerm
drwxr-xr-x  3 joao joao  4096 Ago 14  2013 var
```

Figura 2.11: Ficheiros de um pacote debian

```
Package: apacheds
Version: 2.0.0-M15
Section: devel
Priority: optional
Architecture: amd64
Installed-Size: 11504
Maintainer: Apache Directory Project <dev@directory.apache.org>
Description: LDAP Server
 ApacheDS is an embeddable directory server entirely written in Java, which
 has been certified LDAPv3 compatible by the Open Group. Besides LDAP it
 supports Kerberos 5 and the Change Password Protocol. It has been designed
 to introduce triggers, stored procedures, queues and views to the world of
 LDAP which has lacked these rich constructs.
```

Figura 2.12: Ficheiro *control*

2.8.3 PHP

Como mencionado na secção 2.6, a lógica da API a implementar será desenvolvida na linguagem PHP para manutenção da homogeneidade do sistema em que será integrado. PHP é uma linguagem de programação de servidor procedural e orientada a objetos criada em 1995 e atualmente implementada em milhões de servidores web em todo o mundo. Originalmente criada para desenvolvimento de páginas web dinâmicas, atualmente é também utilizada na lógica de aplicações integradas em servidores web. Encontra-se, aquando do desenvolvimento deste projeto, na versão 5.5.8.

2.8.4 Poster

Devido à limitação do *browser* para pedidos HTTP GET, foi necessário recorrer a uma ferramenta que pudesse enviar pedidos mais customizáveis e que interagissem com todas as funcionalidades da API a ser desenvolvida. A extensão Poster para o *browser* Firefox foi a solução escolhida para envio de pedidos durante a fase de desenvolvimento do projeto pela sua simplicidade e rapidez, ideal para utilização constante. Esta ferramenta permite definir uma vasta gama de características num pedido HTTP direcionado a um determinado URL, como atributos adicionais do cabeçalho, parâmetros do corpo do pedido e credenciais de autenticação básica com recurso a uma interface simples e intuitiva. Com esta ferramenta é também possível utilizar todos os métodos HTTP disponíveis no protocolo, como os métodos POST, PUT, OPTIONS e DELETE entre outros.

2.8.5 Ferramentas named

Named-checkconf e named-checkzone são duas ferramentas Unix que permitem a análise e deteção de erros de sintaxe e tipográficos no ficheiro de configuração do serviço BIND named.conf e erros de semântica nos ficheiros de zonas respetivamente. A primeira permite, além das principais funcionalidades enunciadas,

efetuar um teste de sobrecarga nas zonas *master* identificadas no ficheiro analisado através da ativação da *flag* -z e obscurecimento de chaves que sejam incluídas e sujeitas a análise pela ferramenta através da ativação da flag -x para que o resultado possa ser partilhado sem risco. A ferramenta não apresenta nenhum resultado em caso de sucesso, descriminando os erros identificados durante a análise caso existam. A ferramenta named-checkzone por sua vez é a escolhida para identificar discrepâncias e problemas de integridade nos dados e registos definidos nos ficheiros das zonas a analisar. O maior ponto de customização de uma análise feita com recurso a esta ferramenta prende-se com o grau de exaustão de análise dos registos da zona no que toca à identificação de registos válidos fora da zona escolhida e coerência de registos *glue* entre zonas com relação hierárquica.

Capítulo 3

Solução e Implementação

3.1 Introdução

Após análise das opções apresentadas no capítulo 2 foi feita uma escolha, apresentada e fundamentada na secção 3.1.1. O passo seguinte baseou-se na formulação do esquema concreto da solução tendo em conta o contexto, funcionalidades esperadas e base sobre a qual iria ser implementada. É apresentada e explicada a estrutura desenvolvida da API em 3.1.3 assim como a sua ligação e interação com o sistema IPBrick, descrito na secção 3.1.2. Por fim, é apresentado na secção 3.2 todo o trabalho realizado, sendo analisados de forma mais criteriosa todos os componentes da solução final e fundamentadas todas as escolhas e decisões tomadas no seu desenvolvimento.

3.1.1 Escolha da solução

A principal decisão a tomar no decorrer do projeto, a que moldaria todo o esquema da solução, prendia-se com a arquitetura a adotar. REST ou SOAP? Como enunciado na secção 2.5.3, a escolha não seria simples e direta.
De um lado tínhamos SOAP, a solução maioritariamente escolhida para soluções empresariais, que apresentava um número superior de ferramentas e mais garantias para o estabelecimento de um protocolo fiável entre duas entidades, tudo opções apelativas para o projeto em questão.
Do outro, REST, uma arquitetura mais recente, preparada para o presente e para o futuro, mais flexível em termos de estrutura da mensagem e que funcionaria com qualquer cliente com funcionalidades para executar pedidos HTTP.
Duas opções válidas para a API a desenvolver sendo que a escolha acabou por ser motivada por uma razão que se mostrou relevante para a empresa: perspetivas futuras. Uma API de integração em REST abre portas para o desenvolvimento de toda uma API de comunicação em REST, uma API cujas funcionalidades podem ser acedidas por todo o tipo de dispositivos, sobretudo móveis, com um menor consumo de tráfego e maior velocidade.
Numa altura em que o conhecimento sobre o cliente que estará a usar a API é cada vez mais incerto por aumento do tipo de dispositivos capazes de aceder a serviços através da rede, é necessário que esses serviços acompanhem essa tendência.
A escolha da arquitetura REST implicaria também a opção de utilização de um dos dois formatos de mensagens apresentados em 2.4 e, pela mesma razão, a escolha recaiu sobre a utilização de mensagens formatadas em JSON.

3.1.2 Estrutura da IPBrick

Uma máquina IPBrick funciona de maneira a que todas as modificações necessárias de efetuar no sistema tenham de ser registadas em bases de dados, validadas e só depois implementadas. Além disso, a introdução de dados numa base de dados IPBrick implica uma chamada a uma biblioteca IPBrick que, por sua vez, utiliza uma classe para acesso a funções de manipulação de elementos das tabelas, pelo que todo o percurso de configuração de um sistema pode ser altamente controlado e gerido desde a introdução de dados até à modificação de configurações do sistema. Este esquema combina perfeitamente com o modelo pensado para a API a desenvolver pois também apresenta camadas de abstração bem delineadas e pontos de acesso bem definidos, acentuando algumas das características principais identificadas posteriormente na secção 3.1.3 como é o caso da gestão de recursos.

3.1.2.1 Bases de dados

A solução IPBrick utiliza o sistema PostgreSQL para gestão das suas bases de dados e correspondentes tabelas sendo que a API em questão apenas irá fazer uso da base de dados responsáveis por armazenamento das configurações do sistema. As configurações presentes nesta base de dados aquando a ordem de aplicação de modificações serão as aplicadas a todo o sistema, estando a manipulação de ficheiros, execução de comandos e validação de alguns registos à responsabilidade do *software* IPBrick.

3.1.2.2 Classes de acesso à base de dados

Como enunciado anteriormente, a modificação de qualquer tabela nunca é direta mas sim feita com recurso a classes com funções próprias. Esta camada garante não só a tradução de parâmetros em *queries* como também implementa, em certos casos, verificação desses mesmos parâmetros. Desta maneira é também limitada a utilização e criação de *queries* a uma camada controlada pela IPBrick, garantindo segurança contra possíveis abusos como é o caso de tentativas de *SQL injection*. Estas funções podem também manipular uma tabela de alterações que regista os serviços que sofreram modificações e que, numa aplicação de configurações, terão de ser revistos, garantindo uma gestão de recursos mais eficaz por não necessidade de revisão de todos os serviços aquando a introdução de modificações no sistema. São asseguradas pela IPBrick classes de acesso e manipulação para todas as tabelas existentes.

3.1.2.3 Bibliotecas

As bibliotecas IPBrick serão o ponto de comunicação entre o sistema IPBrick e a API a desenvolver. São responsáveis pela gestão das chamadas à classes de acesso e estão divididas por serviços, facilitando então a correspondência entre uma biblioteca IPBrick e um controlador de serviço da API de integração. Esta camada da estrutura é maioritariamente responsável pela análise dos parâmetros fornecidos, reestruturação desses mesmos parâmetros para envio numa chamada a uma classe de acesso, verificação do estado do sistema e obtenção de informações relevantes. É nestas bibliotecas que toda a lógica se encontra e onde é feita a maior filtragem de pedidos com valores corretos ou inválidos.

3.1.3 Estrutura da API a desenvolver

A API de integração desenvolvida segue uma estrutura elaborada com base no modelo MVC apresentado na secção 2.7 sendo um esquema funcional disponível na figura 4.28. Através deste modelo modular é possível garantir separação de responsabilidades e definição de zonas de acesso distintas mantendo um alto nível de

escalabilidade e eficácia. O seu único ponto de contato exterior (a interface) permite um maior controlo de acesso às funcionalidades oferecidas através da análise de pedidos e gestão de recursos por triagem e encaminhamento adequado dos mesmos através dos módulos específicos para o serviço requisitado. O acesso ao serviços do sistema IPBrick ocorre através das bibliotecas fornecidas e carregadas pelo respetivo controlador. Este controlador tem a função de analisar o pedido identificando a sua intenção e parâmetros, intercedendo sobre as funções necessárias de maneira a responder ao pedido. São também registados os acessos (bem ou mal sucedidos) à API com informações pertinentes para os descrever assim como todos os erros que possam ocorrer durante a sua utilização para posterior análise. A escalabilidade deste sistema prende-se com a distribuição de tarefas por diferentes módulos. Com a adição de um novo serviço apenas se torna necessária a criação de um controlador específico que, por si só, já herda funcionalidades de um controlador geral. A necessidade de fornecer suporte para um novo formato de mensagem de saída apenas compreende a criação de um modelador de respostas adequado que, por sua vez, também herda algumas das suas funcionalidades e estrutura de um modelador geral já existente. A modificação do controlo de acesso também apenas afeta o módulo responsável por essa função, assim como a gestão de mensagens de erros e traduções. Passamos então à descrição das funcionalidades, características e estrutura dos diferentes módulos que compõem a API a desenvolver.

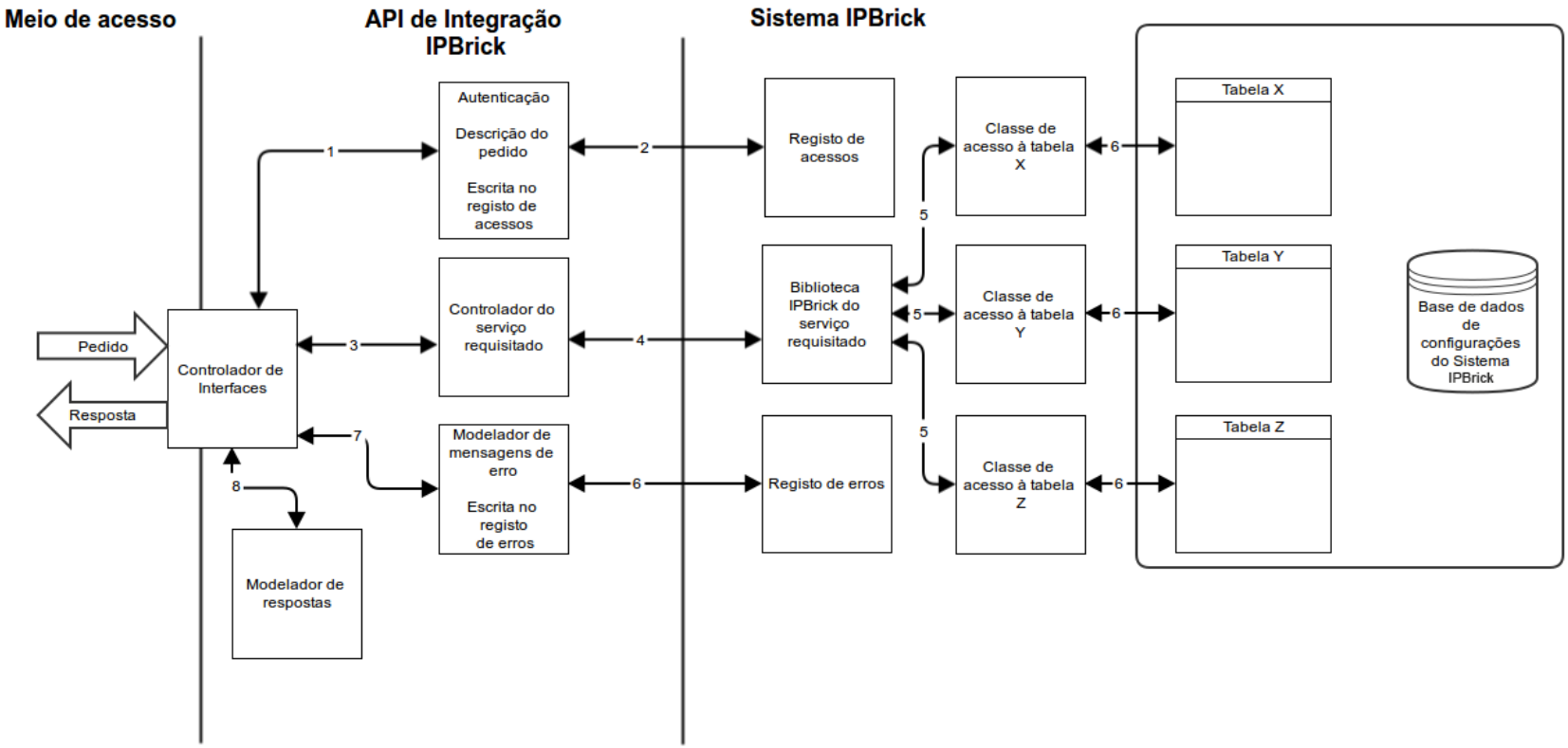

Figura 3.1: Diagrama de fluxo da API

3.1.3.1 Controlador de interfaces

O controlador de interfaces compõe o primeiro e único ponto de acesso de um pedido externo à API e, por consequinte, ao sistema. Tem como objetivo a atribuição de tarefas aos diferentes módulos, gestão dos mesmos e retorno das respostas. A utilização de um único ponto de entrada garante várias vantagens. Por um lado permite a identificação unívoca do acesso à API. Por outro, permite um controlo mais eficaz dos acessos com garantia de que todos os pedidos são processados da mesma maneira oferecendo ainda mais um ponto de escalabilidade visto a necessidade de adição de uma nova etapa de processamento de um pedido apenas compreende a ligação de um novo módulo e direcionamento do pedido ou parte dele para o mesmo.

3.1.3.2 Interface de acesso

Todos os pedidos que são enviados para o controlador de interfaces são, por sua vez, redirecionados para este módulo. É nesta etapa que o pedido é analisado e escrutinado de maneira a serem identificados os seus diferentes componentes que serão depois utilizados, como é o caso do URL, parâmetros fornecidos através do URL, método do pedido e parâmetros do corpo do pedido entre outros. É também nesta fase que ocorre uma triagem dos acessos, sendo rejeitados pedidos que não estejam de acordo com os critérios de autenticação e/ou permissão, de maneira a salvaguardar a integridade do sistema e garantir uma gestão de recursos mais eficaz ao rejeitar pedidos numa fase ainda inicial do processo. Por fim, mesmo no caso de serem rejeitados, todos os acessos são então registados internamente para posterior análise e monitorização dos mesmos.

Autenticação e Permissão
O controlo de acesso às funcionalidades da API e a segurança deste acesso é feito com recurso a 4 características implementadas tendo sido discutidos na secção 2.5.3.6 com maior detalhe. Na escolha de uma solução foram tomados em conta os seguintes fatores:

(a) Toda a comunicação com a interface IPBrick aquando a realização deste projeto era efetuada com recurso exclusivo ao protocolo HTTPS;

(b) Sendo uma API inicialmente projetada para utilização local e ambiente controlado com risco baixo de intrusão, optou-se pela redução de complexidade do mecanismo de autenticação em prol de velocidade na comunicação e da implementação;

(c) O sistema IPBrick já se encontra estruturado numa ideologia de controlo de acesso por credenciais;

(d) É expectável a necessidade de definição de acesso a conjuntos de funcionalidades a grupos de utilizadores distintos.

A utilização do protocolo HTTPS na comunicação torna-se então um requisito de implementação natural no sistema. Já a escolha de *Basic Authentication* provém da identificação de um mecanismo simples de controlo de acesso por meio de credenciais. A implementação de uma chave API garante não só uma segunda camada de controlo de acesso como também apresenta uma maneira eficaz e rápida de implementação de grupos de acesso. Por fim, a identificação da origem do pedido permite adicionar ou eliminar camadas de proteção a determinados pedidos de maneira a acelerar o processo de comunicação.

1. HTTPS A necessidade do estabelecimento de um túnel SSL sobre o protocolo HTTP nas comunicações

efetuadas entre o cliente e o sistema assegura a proteção das mensagens contra monitorização do tráfego. Esta característica apenas atua na vertente de segurança da comunicação e não no controlo de acesso por parte do cliente.

2. Credenciais - *Basic Authentication* O recurso a credenciais de acesso transmitidas por *Basic Authentication* (ver secção 2.5.3.6) permite não só validar a autorização de um cliente no acesso a recursos da API como também identificar o remetente do pedido. É importante salientar que, sendo a arquitetura REST independente do estado da aplicação e do sistema, esta verificação é feita por pedido e em todos os pedidos, pelo que a mesma máquina pode enviar mensagens com credenciais distintas. O nome de utilizador usado fica associado a cada acesso nos registos de acesso para facilitação da sua análise.

3. Permissão - chave API
É prática comum a utilização de um elemento enviado aquando um pedido, normalmente via URL. Este elemento que identificamos como uma chave API permite o controlo da utilização da API como uma chave de acesso comum, sendo que é possível associar uma determinada chave tanto a um grupo de utilizadores como a um grupo de funcionalidades.

4. Localização da origem do pedido
Dado a comunicação ser feita por meio de protocolo HTTP é possível o acesso às funcionalidades da API serem acedidas remotamente e não localmente. É, no entanto, possível identificar a origem de cada pedido e efetuar a comparação com a localização dos serviços inferindo tratar-se de um pedido local ou remoto e podendo ser controlado o seu acesso com base nessa informação.

3.1.3.3 Controlador de serviço

Foi criado um controlador para cada serviço disponibilizado através da API sendo este modulo o ponto de comunicação com as bibliotecas do sistema IPBrick. Um controlador tem como objetivo identificar a intenção do pedido e efetuar as chamadas a funções das bibliotecas IPBrick com os parâmetros que lhe foram passados pelo módulo anterior de maneira a responder a esse pedido. A estrutura da API funciona de tal maneira que o URL de destino do pedido HTTP identifica o controlador que deve ser chamado sendo o verbo do pedido identificador do método que será utilizado. Cada controlador de serviço tem definidos um máximo de 4 métodos, um por cada ação HTTP disponível (GET, POST, PUT, DELETE), e um método relativo ao verbo OPTIONS herdado de um controlador geral sendo que este último permite a identificação das ação disponíveis em determinado serviço. Esta dinâmica permite que, por cada URL, seja possível efetuar até 5 ações a um determinado recurso e que todos os tipos de recursos sejam apenas acedidos através de um controlador específico. Isto traz benefícios em termos de eficácia visto que apenas se torna necessário carregar ficheiros essenciais para a manipulação de um tipo de recurso por pedido. Será importante denotar que o fato de as bibliotecas IPBrick definirem funções seguindo a filosofia CRUD faz com que a relação método do controlador-função da biblioteca seja direta pois, como explicado na secção 2.5.3.1, existe uma correspondência entre as ações CRUD e os verbos HTTP principais. É também neste módulo que é feita uma segunda triagem por análise de parâmetros do pedido. São evitadas assim diversas chamadas a funções das bibliotecas que, por sua vez, se poderiam traduzir em acessos desnecessários a bases de dados em situações em que o pedido não pode ser satisfeito por falta ou disponibilização incorreta de parâmetros. No caso de o pedido satisfazer as condições impostas pelo controlador, é formulada uma chamada à função necessária com parâmetros organizados com base na informação fornecida pelo pedido. Essa chamada irá retornar dois valores: o valor resultado da função e o valor de erro. No caso da chamada ser mal-sucedida,

o primeiro valor será sempre igual a 1 e é especificado o ou os erros ocorridos através de um ou mais códigos de erro no segundo valor. Com estes dados o controlador formula uma resposta final a enviar para o controlador de interfaces. É importante que todos os controladores estruturem as suas respostas da mesma maneira e esta seja independente do conteúdo da resposta. Isto garante a modularidade e escalabilidade do sistema visto que esta informação pode, em módulos seguintes, ser tratada toda da mesma maneira como veremos que acontece nesta situação.

1) Controlador principal

A necessidade de garantir um controlador principal comum a todos os controladores de serviços advém da importância de garantir que todos possuam características comuns que possam ser facilmente implementadas. Esta estrutura permite reenforçar a utilização do modelo MVC com especialização dos componentes. Na classe que define o controlador principal são incluídos ficheiros necessários a todos os serviços, ligação à base de dados, um construtor comum para identificação do método a ser utilizado, definição do método referente à ação OPTIONS e o método de verificação dos parâmetros fornecidos utilizado na triagem ao nível dos controladores.

2) Estrutura das respostas

Como referido previamente, existe a necessidade de formalizar uma estrutura única e comum, independente do conteúdo, para as respostas fornecidas pelos controladores de serviço ao controlador de interfaces. Foi então especificada um formato de resposta formada por dois elementos. O primeiro é utilizado para enviar valores numéricos (“code”) e o segundo para informação complementar (“message”). Esta estrutura foi também adotada para transmissão da resposta final ao pedido. Os valores numéricos têm duas utilizações principais permitindo especificar o estado da resposta e, caso necessário, os erros ocorridos. De maneira a garantir o máximo de informação por resposta este valor é estruturado num vetor. Estes valores são analisado no controlador de interfaces e direcionados para a interface de gestão de erros caso sejam identificados códigos referentes a problemas ocorridos. A informação complementar serve para armazenamento dos dados pedidos em situações bem-sucedidas e para transmissão de valores relevantes relativos aos erros ocorridos. De maneira a garantir que não são tratadas como contendo erros respostas de chamadas bem-sucedidas, estas têm de retornar obrigatoriamente apenas o valor 200 ou 201 no campo “code” sendo o segundo utilizado para pedidos POST. É preciso reforçar o fato de apenas respostas com valor único de 200 ou 201 não serem redirecionados para a interface de gestão de erros, pois esta também é responsável pela gestão de avisos, avisos estes que podem não constituir condição suficiente para identificar um pedido mal-sucedido.

3.1.3.4 Interface de gestão de erros

Como mencionado na secção 3.1.3.3, na ocorrência de erros ou avisos, as respostas são redirecionadas para um módulo que assegura a tradução dos códigos de erro em texto informativo que pode ser utilizado para depuração e identificação das causas de pedidos mal-sucedidos. Para uma formulação mais adequada e específica do erro ocorrido, podem ser armazenados no campo “message” strings extra. O então texto formulado é guardado no campo “message”, sendo este valor constituinte da resposta final juntamente com o campo “code” original. Esta resposta é, além de transmitida à fonte do pedido, guardada num ficheiro de registo no sistema.

3.1.3.5 Modelador de respostas

Após definido o conteúdo da resposta e a sua estrutura, é necessária a sua formatação num dos formatos de mensagem disponíveis. Seguindo uma estrutura MVC, o componente responsável por esta função é o View. No entanto foi utilizada a mesma abordagem a este componente que a utilizada nos controladores, mantendo um componente principal e criando módulo especializados que herdam as suas características mas são chamados em situações distintas. A situação atual da API desenvolvida garante respostas no formato em que são aceites os pedidos, JSON, pelo que só se encontra desenvolvido um View especializado. No entanto é possível a implementação de um módulo que permita respostas em outros formatos graças à estrutura da API. Este módulo tem como função, além da formatação das respostas em formato reconhecível pela origem do pedido, a tradução de nomes de parâmetros de maneira a assegurar a homogenidade da comunicação.

3.2 Desenvolvimento

As funcionalidades disponíveis nos diversos serviços podem ser divididas em 2 tipos: listagem de recursos e manipulação de recursos. As primeiras providenciam respostas que incluem representações não completas de recursos de um determinado tipo e que, em certos casos, podem ser filtrados por características especificadas. São respostas obtidas somente a pedidos GET, por norma, ao URI base do serviço. Têm como principal objetivo fornecer ao cliente informação sobre a indexação dos recursos de maneira a este poder identificar um elemento específico, quer para posterior manipulação, quer somente para verificação da sua existência. As funcionalidades que visam a manipulação de recursos englobam, por regra, obtenção de todos os seus atributos registados no sistema, edição desses mesmos atributos e criação ou eliminação de um recurso de um determinado tipo. Estas acções abrangem toda a gama de tipos de pedidos HTTP pelo que podem exigir informação transmitida através do corpo do pedido, tendo essa informação de ser fornecida no formato de mensagem aceite pela API (neste caso, como referido anteriormente, JSON).

3.2.1 Serviços

A opção de agrupar os serviços oferecidos pelo sistema IPBrick advém da existência de uma estrutura comum na lógica de todos os controladores de serviços deste género. Tendo sido seguido um padrão base no desenvolvimento da solução faz então sentido analisa-lo e posteriormente identificar as especificações de cada serviço individualmente.

3.2.1.1 Funcionalidades

Excetuando o serviço de gestão do processo de instalação abordado posteriormente numa secção individual, todos os serviços e sub-serviços providenciados pela API foram estruturados de maneira a funcionarem de maneira análoga. A referência a sub-serviço advém da divisão do serviço geral de gestão de registos DNS em sub-serviços responsáveis pela gestão de registos de diferentes tipos assim como um sub-serviço extra de gestão de zonas DNS. Sendo a arquitetura REST adotada baseada em recursos e na representação de cada um desses recursos, faz sentido a divisão de acesso a grupos de recursos ou a recursos individuais. Os grupos aqui referidos englobam desde a totalidade de recursos de um determinado tipo até a um sub-conjunto dessa totalidade com caraterísticas comuns especificadas de antemão. Sendo já a estrutura do sistema IPBrick dividida em serviços, foi então adaptada essa divisão na API a desenvolver e aplicado o principio referido previamente a cada um destes serviços. Isto traz grandes vantagens no sentido de:

- Acesso otimizado a um coletivo de recursos de um serviço ou sub-serviço

- Fácil e intuitiva elaboração de um URI de acesso a um recurso individual de um serviço ou sub-serviço
- Especificação e identificação de um recurso dentro de um serviço ou sub-serviço através do seu URI de acesso
- Generalização de métodos de acesso e gestão de um serviço ou sub-serviço

Este último ponto advém da utilização dos verbos HTTP na identificação de métodos de acesso, garantindo uma homogeneidade e padrão comum de utilização dos serviços.
Comecemos então por uma análise mais detalhada de cada serviço fornecido.

Serviço de Gestão de *Virtual Hosts*

Web Services		GET	POST	PUT	DELETE
/vhost	Gestão de Virtual Hosts	Lista de Virtual Hosts	Criação de VH	-	-
/vhost/<idvhost>		Informação sobre Virtual Host	-	Modificação de VH	Eliminação de VH
/vhost/changes /<idbugfixes>		Lista de identificadores dos Virtual Hosts criados pela aplicação <idbugfixes>	-	-	-

Figura 3.2: Funcionalidades do serviço de gestão de *virtual hosts*

Identificando um *virtual host* como o recurso básico deste serviço, é possível o acesso a todos os *virtual hosts* registados no sistema através do URI genérico do serviço, manuseamento de um *virtual host* através da utilização do identificador unívoco desse recurso no URI e, por fim, identificação de todos os recursos deste tipo criados por uma determinada aplicação com base no seu identificador. Esta funcionalidade apresenta-se útil no processo de desinstalação de uma aplicação, permitindo listar os identificadores dos recursos a remover.

Serviço de Gestão de regras de *firewall*

Web Services		GET	POST	PUT	DELETE
/firewall	Gestão de Firewall	Lista de regras	Adição de regra na Firewall	-	-
/firewall?<filtro>= <valor>		Lista de regras filtradas	-	-	-
/firewall/<idfirewall>		Informação sobre regra	-	Modificação de regra na Firewall	Eliminação de regra na Firewall
/firewall/changes /<idbugfixes>		Lista de identificadores das regras criadas pela aplicação <idbugfixes>	-	-	-

Figura 3.3: Funcionalidades do serviço de gestão de regras de *firewall*

O serviço de gestão de regras de *firewall* providencia a adição e listagem das regras existentes com ou sem filtragem aplicada através do URI genérico de acesso ao serviço: /firewall. Com a utilização de um identificador no URI é possível obter informações acerca da regra definida por esse identificador, aplicar modificações nessa mesma regra ou removê-la da *firewall* do sistema com a utilização dos pedidos do tipo GET, PUT e DELETE respetivamente. Analogamente ao serviço de gestão de *virtual hosts*, é também providenciado o serviço de obtenção de identificadores de recursos criados por uma dada aplicação através do URI /firewall/changes.

Serviço de Gestão de contas de utilizadores

Web Services		GET	POST	PUT	DELETE
/user	Gestão de Utilizadores	Lista de utilizadores	Criação de Utilizador	-	-
/user?login=<login>		Informação sobre utilizador com login=<login>	-	-	-
/user?<parameter>=<parameter_value>&...		Informação sobre utilizador com os parâmetros definidos (1)	-	-	-
/user/<iduser>		Informação sobre utilizador	-	Modificação de dados de Utilizador	Eliminação de Utilizador
/user/changes/<idbugfixes>		Lista de identificadores dos utilizadores criados pela aplicação <idbugfixes>	-	-	-

Figura 3.4: Funcionalidades do serviço de gestão de contas de utilizadores

O serviço de gestão de contas de utilizadores permite a configuração de uma nova conta, assim como a listagem de todas as contas existentes no sistema podendo esta lista também ser sujeita a uma filtragem por valores de parâmetros. Uma referência terá de ser feita a este ponto, visto o processo de filtragem não permitir a definição de todos os parâmetros disponíveis através do recurso, razão pela qual é aconselhado a utilização apenas dos parâmetros "login"e "name"e utilização de carateres codificados para URL. Mais uma vez são apresentadas as funcionalidades de manuseamento de um recurso como a sua edição, remoção e obtenção dos seus detalhes através dos métodos HTTP apresentados e o sub-serviço de listagem de recursos criados via aplicação.

Serviço de Gestão de registos e zonas DNS

Recursos relacionados com DNS como registos e zonas encontram-se acessíveis através do URI /dns. São então divididos nos diferentes tipos de registos e num sub-serviço de gestão de zonas, fornecendo cada um destes as funcionalidades dos serviços apresentados até agora, desde a listagem com ou sem filtragem por zona ou aplicação e adição, modificação e eliminação de registos/zonas. No entanto, além da divisão em sub-serviços, é possível identificar mais duas peculiaridade neste serviço: gestão conjunta de registos A e PTR e validação dos ficheiros DNS. A primeira prende-se com a ligação intrínseca dos dois tipos de registos, oferecendo a possibilidade de automatizar o processo de configuração de um registo PTR aquando a criação, modificação ou eliminação de um registo A, poupando a necessidade de acesso a dois sub-serviços. Já a segunda funcionalidade apresenta a possibilidade de validação manual de modificações feitas ao serviço DNS através do URI /dns/check e identificação do domínio a validar.

Serviço de Gestão de rotas SMTP

Embora não incluído na análise de requisitos inicial, durante o processo de implementação e teste da implementação foi identificada a necessidade de controlo de configurações do servidor de correio eletrónico, principalmente a gestão de rotas SMTP. Esta funcionalidade foi então adicionada ao também criado serviço de gestão de configurações de correio eletrónico. Embora atualmente limitado em funcionalidades, a criação deste serviço inicia um processo inclusão de um serviço com várias vertentes passíveis de serem exploradas como a definição de domínios de entrega local, controlo de filas de espera e configuração de listas de e-mail.

Web Services		GET	POST	PUT	DELETE
/dns/mx	Gestão de registos MX	Lista de registos MX	Criação de Registo MX	-	-
/dns/mx?idzone=<idzone>		Lista de registos MX da zona <idzone>	-	-	-
/dns/mx/<iddns_mx>		Informação de registo MX	-	Modificação de registo MX	Eliminação de registo MX
/dns/ns	Gestão de registos NS	Lista de registos NS	Criação de Registo NS	-	-
/dns/ns?idzone=<idzone>		Lista de registos NS da zona <idzone>	-	-	-
/dns/ns/<iddns_ns>		Informação de registo NS	-	Modificação de registo NS	Eliminação de registo NS
/dns/txt	Gestão de registos TXT	Lista de registos TXT	Criação de Registo TXT	-	-
/dns/txt?idzone=<idzone>		Lista de registos TXT da zona <idzone>	-	-	-
/dns/txt/<iddns_txt>		Informação de registo TXT	-	Modificação de registo TXT	Eliminação de registo TXT
/dns/srv	Gestão de registos SRV	Lista de registos SRV	Criação de Registo SRV	-	-
/dns/srv?idzone=<idzone>		Lista de registos SRV da zona <idzone>	-	-	-
/dns/srv/<iddns_srv>		Informação de registo SRV	-	Modificação de registo SRV	Eliminação de registo SRV (4)
/dns/a	Gestão de registos A e PTR	Lista de registo A e PTR	Criação de Registo A e PTR	-	-
/dns/a?idzone=<idzone>		Lista de registos A e PTR da zona <idzone>	-	-	-
/dns/a/<iddns_a>		Informação de registo A e PTR	-	Modificação de registo A e PTR	Eliminação de registo A e PTR (4)
/dns/zone	Gestão de zonas DNS	Lista de zonas DNS	Criação de zona DNS	-	-
/dns/zone/<iddns_zone>		Informação de zona DNS	-	Modificação de zona DNS	Eliminação de zona DNS
/dns/zone/changes/<idbugfixes>		Lista de identificadores das zonas criadas pela aplicação <idbugfixes>	-	-	-
/dns/check/<domain>	Validação dos registos DNS	Valida os ficheiros BIND para o domínio <domain>	-	-	-

Figura 3.5: Funcionalidades do serviço de gestão de registos e zonas DNS

Web Services		GET	POST	PUT	DELETE
/mail/smtproute	Gestão de rotas SMTP	Lista de rotas SMTP	Criação de rota SMPT	-	-
/mail/smtproute/<idsmtproute>		Informação sobre rota SMTP	-	Modificação de rota SMTP	Eliminação de rota SMTP

Figura 3.6: Funcionalidades do serviço de gestão de rotas SMTP

Serviço de Gestão de entradas no registo de *bugfixes*

Web Services		GET	POST	PUT	DELETE
/bugfix	Gestão de entradas na lista de bugfixes	Lista de entradas na lista de bugfixes	Criação de uma entrada	-	-
/bugfix /<idbugfixes>		Infomação sobre entrada	-	Modificação de uma entrada	Eliminação de uma entrada

Figura 3.7: Funcionalidades do serviço de gestão de entradas no registo de *bugfixes*

Uma necessidade referida na análise de requisitos apresentada (secção 1.5) visa o registo dos detalhes da aplicação e da instalação no sistema após integração bem-sucedida, tais como uma descrição do pacote e data de instalação entre outros. Desta maneira é possível manter atualizada uma lista de pacotes instalados no sistema, tanto para conhecimento do cliente através da interface gráfica disponibilizada pela IPBrick como para a gestão desses mesmos pacotes por parte da API e do sistema. É importante referir que a introdução de um registo na tabela de *bugfixes* não se encontra sujeita a uma análise de compatibilidade como a efetuada no inicio de uma instalação, razão pela qual é sugerida a sua utilização apenas no final do processo de instalação e com a certeza do sucesso de todo o processo. Este serviço garante não só acesso a uma lista registos de aplicações instaladas no sistema como permite, considerando um registo como um recurso do sistema, o seu manuseamento (edição, eliminação, criação e obtenção de detalhes).

3.2.1.2 Estrutura

Como referido anteriormente, todos os controladores de serviços fornecidos pelo sistema IPBrick foram estruturados de maneira a seguirem um padrão comum. O processo de execução de um controlador é então o seguinte:

1. Verificação do URL do pedido
2. Redirecionamento do pedido para o sub-serviço correto
3. Validação dos parâmetros específicos da API
4. Validação da sintaxe dos parâmetros do pedido (opcional)
5. Execução da chamada à biblioteca
6. Validação da resposta

Analisemos então a importância desta ordem de acontecimentos aquando o envio de um pedido para um controlador de serviço. O primeiro ponto serve como triagem, identificando pedido inválidos mesmo tendo sido possível o redirecionamento para um controlador. Respostas de erro a pedidos que sejam invalidados neste ponto envolvem a indicação da sintaxe correta de acesso ao serviço ou simplesmente a indicação de pedido mal-formulado. O segundo ponto apenas é implementado devido à existência de sub-serviços incluidos nos diversos controladores desde a listagem de identificadores de recursos criados por uma aplicação de um determinado tipo até sub-serviços completos como é caso dos vários tipos de registos acDNS. Através do URL do pedido, esta etapa identifica o sub-serviço correspondente (caso seja necessário) e reencaminha o pedido para um ponto de tratamento mais adequado. É importante a correta definição das etapas de validação dos parâmetros do pedido. Embora tenha sido definida uma atribuição bem delimitada de tarefas no que toca a este tema para a solução desenvolvida em relação ao que é feito nas bibliotecas IPBrick, surgiu a necessidade de informar o cliente aquando a incorreta definição dos parâmetros mesmo estes sendo

analisados e validados com mais detalhe nas bibliotecas. Estes pontos de validação surgem então mais como um método de envio de informação relevante quanto à invalidade de um pedido por não-existência de certos parâmetros do que como uma filtragem por análise dos mesmos. A identificação como caráter opcional do ponto 4 surge por apenas necessidade de execução em tipos de pedido que enviem corpo de mensagem. A etapa de execução da chamada aparece como a ligação entre a API e o sistema IPBrick já implementado, sendo identificada a função mais apropriada para tratamento de um pedido e enviados os parâmetros extraídos desse mesmo pedido de maneira a serem utilizados pela função. Ao obter um retorno por parte da função executada, a API pode então validar o pedido como válido ou inválido, bem-sucedido ou mal-sucedido e elaborar uma resposta com os erros ocorridos quer no âmbito da função do sistema IPBrick, quer no âmbito da API de integração.

3.2.2 Serviço de Instalação

Dada a natureza da utilização da API em processos de instalação e tendo em consideração a necessidade de implementação de um conjunto de validações e alterações iniciais em todos estes foi criado um serviço específico da solução desenvolvida para auxílio, controlo e automatização do processo. O controlador deste serviço obedece à estrutura especificada na secção 3.2.1 no entanto adiciona algumas funcionalidades e etapas necessárias de serem executadas apenas pela API e não por bibliotecas específicas.

3.2.2.1 Funcionalidades

Foram atribuídas ao serviço de gestão da instalação as funcionalidades relacionadas com os seguintes pontos: 1. Controlo de concorrência; 2. Validação dos parâmetros da aplicação; 3. Gestão da base de dados de configurações predefinidas; 4. Controlo de modificação inválida de recursos.

O ponto 1 visa garantir o acesso exclusivo aos recursos do sistema por parte de apenas um processo de instalação evitando problemas de concorrência e possíveis incompatibilidades. Este controlo é feito de maneira simples, mantendo um ficheiro temporário durante o processo de instalação e removendo-o aquando a sua finalização. Este mecanismo permite também controlar a regularização do processo, obrigando não só uma aplicação a efetuar uma chamada de finalização de instalação quando termina mas também identificando um processo não-finalizado em curso. Outro mecanismos poderiam ter sido utilizados para esse efeito, contudo o próprio sistema IPBrick apresenta funcionalidades mais completas que garantem exclusividade durante a instalação de um pacote, razão pela qual se optou pela opção mais simples que simplesmente adiciona um segundo ponto de controlo. Foi também analisada a possibilidade de manter informação relativa ao processo de instalação no ficheiro temporário criado, adicionando a possibilidade de retoma da instalação ou identificação da aplicação a ser instalada. No entanto tais funcionalidades não se revelavam prioritárias, embora tivessem sido tomadas em consideração para implementação futura.

O processo de validação das características de uma aplicação apresenta-se de enorme importância no contexto da sua instalação. É neste ponto que é contemplada a compatibilidade da aplicação com o sistema em causa, identificando dependências e possíveis conflitos entre pacotes, versões mínimas necessárias e dupla instalação de uma aplicação. Qualquer uma desta situações ao não ser analisada poderá resultar numa instalação completa e sem erros mas que introduza possíveis incongruências e torne o sistema instável, situação não desejável.

Os pontos 3 e 4 encontram-se relacionados. A necessidade de manter configurações predefinidas atualizadas num sistema IPBrick obriga a que todas as alterações efetuadas durante um processo de instalação sejam registadas numa base de dados temporária e, posteriormente, sejam guardadas num ficheiro interno.

Isto implica não só a execução de modificações em duas bases de dados como a implementação e armazenamento de uma delas no início e final do processo de instalação respetivamente. A obrigatoriedade de execução de modificações nas duas bases de dados traduz-se na restrição de apenas garantir permissão de manipulação de recursos em situações de operacionalidade das duas, nomeadamente em processos de instalação.

3.2.2.2 Estrutura

Dadas as adicionais tarefas deste serviço exclusivamente implementado na API, são apresentados na figura 3.8 e 3.9 diagramas de fluxo detalhados com mensagens de resposta do processo de início de instalação e finalização de instalação respetivamente.

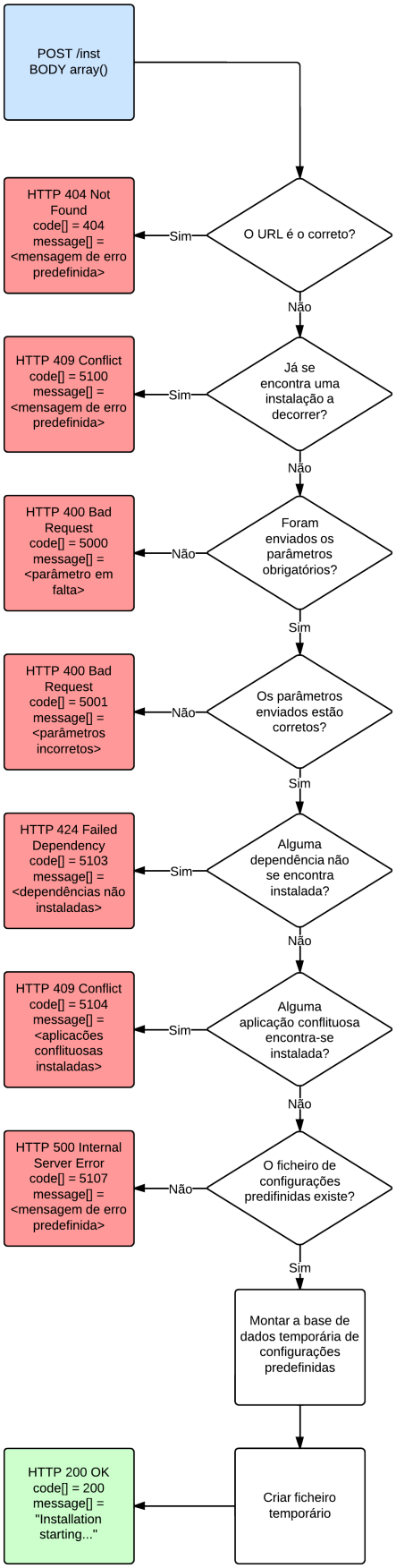

Figura 3.8: Diagrama de fluxo de inicio do processo de uma instalação

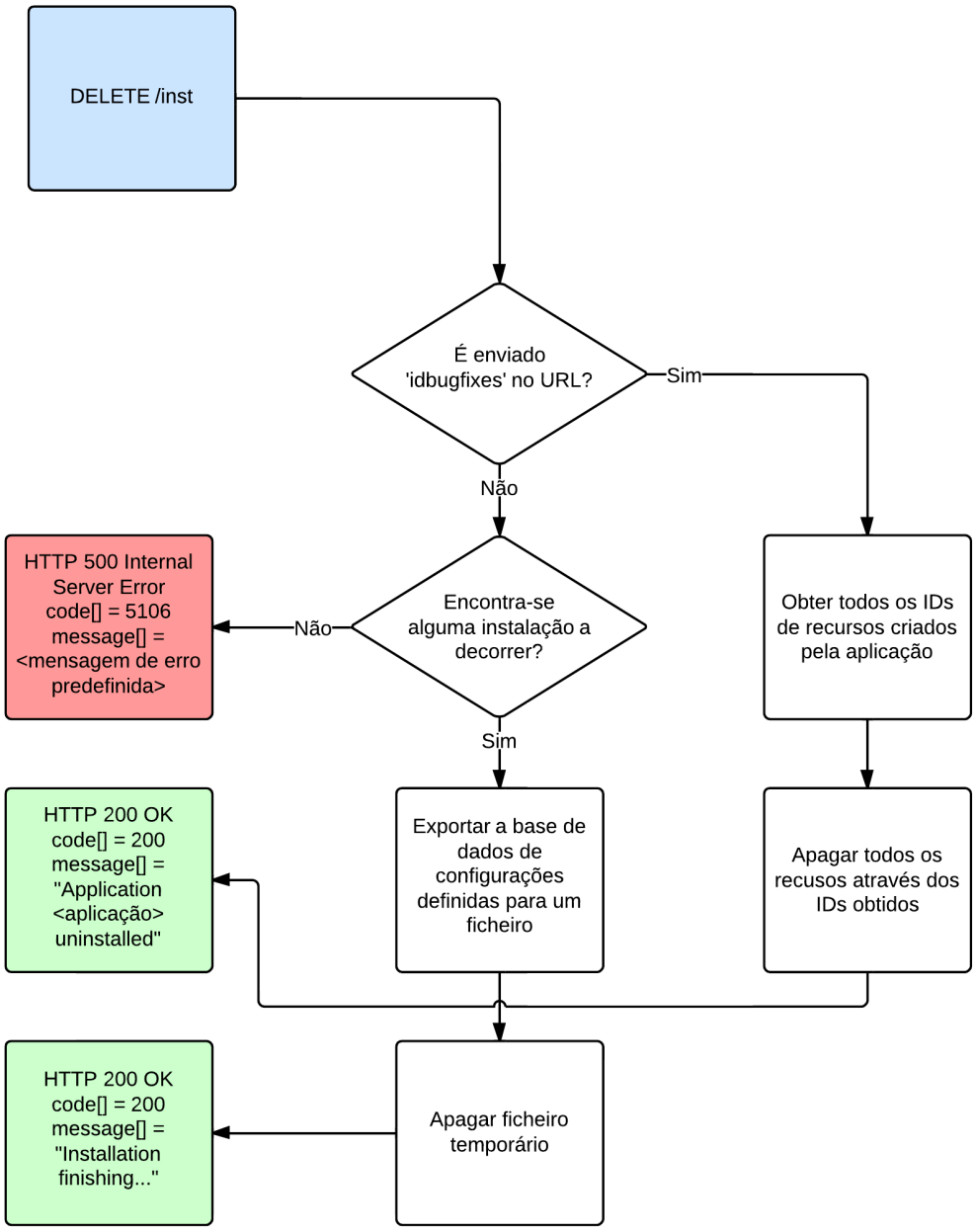

Figura 3.9: Diagrama de fluxo da finalização do processo de uma instalação

Capítulo 4

Validação da Solução

4.1 Procedimentos de Validação

Para validação da solução produzida recorreu-se a um conjunto de testes às suas caraterísticas e funcionalidades. Com a ajuda das ferramentas apresentadas na secção 2.8 foi possível a execução e, em certos casos, automatização de conjuntos de testes procedidos de uma análise aos resultados obtidos. Numa primeira fase foram efetuados testes de pedidos simples com vista a validar pontos específicos da API como é o caso do sistema de autenticação e controlo de erros entre outros. Para estes testes foi utilizada a ferramenta SOAP UI pois esta permite a visualização do pedido executado na íntegra tanto dos cabeçalhos da resposta ao pedido como do seu conteúdo já formatado em JSON o que acelera o processo de análise. A estrutura, os resultados e uma análise dos resultados deste tipo de testes é apresentado na secção 4.2. Utilizando algumas funcionalidades da mesma ferramenta foi possível organizar casos de teste formados por conjuntos de pedidos ordenados com respostas automaticamente validadas segundo condições previamente definidas. Esta organização permitiu também a simulação de diferentes padrões de processos de instalação simultâneos e/ou consecutivos, analisando o impacto de sobrecarga e processamento contínuo na solução desenvolvida. Aproveitando também o poder de automatização de tarefas pelo programa SOAP UI procedeu-se à execução de testes de segurança a elementos de *input* no sistema numa tentativa de identificar possíveis vulnerabilidades da API de integração. Mais informações sobre esta fase de testes são descriminadas na secção 4.4. A última fase de testes consistiu na integração de uma aplicação no sistema através da solução desenvolvida, começando por um pacote-exemplo desenvolvido explicitamente para teste de todas as funcionalidades da API, avançando para um pacote IPBrick mais simples e terminando numa aplicação IPBrick mais completa e pública, tendo este teste o objetivo de viabilizar o produto desenvolvido como uma versão inicial de uma solução comercial demonstrando as suas vantagens em relação à sua integração tradicional. Todo o processo de integração das aplicações assim como a sua estrutura e resultados obtidos encontram-se descritos na secção 4.5.

4.2 Demonstração de Funcionamento

O primeiro teste procurou identificar o funcionamento adequado do controlo de permissão através da utilização de uma chave API.

Como é possível identificar na figura 4.1 nas secções 1 e 2, foi atribuído um valor diferente do esperado como chave e enviado através do URL do pedido. Do lado esquerdo da figura temos acesso ao pedido

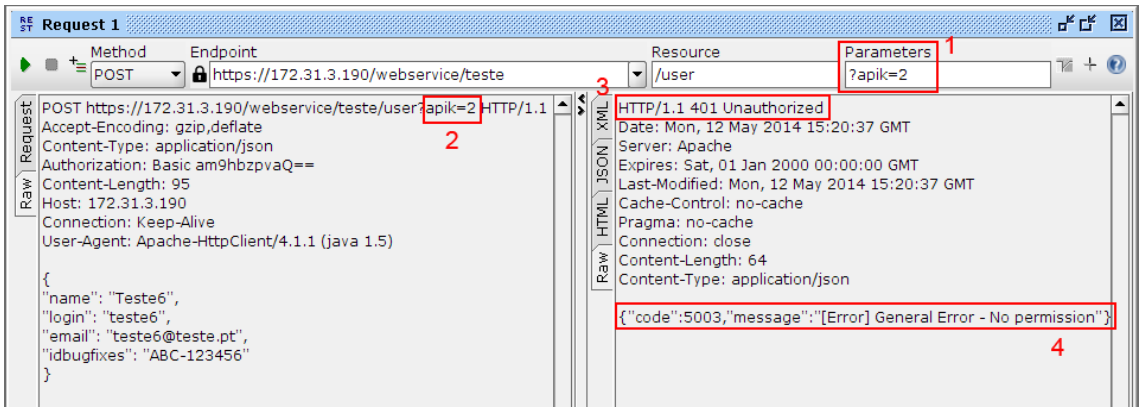

Figura 4.1: Teste de validação do controlo de permissão

na íntegra formado e enviado enquanto que do lado direito é-nos apresentada a resposta proveniente da API. Após identificação do uso incorreto da chave API, o sistema responde com um código HTTP "*401 Unauthorized*"como pretendido, identificando a falta de permissão de acesso por parte do utilizador que efetuou o pedido. No conteúdo da resposta é salientada essa falta de permissão através de um código de erro específico da API e uma mensagem representativa do erro ocorrido.

Além do controlo de acesso através da chave API também é efetuado o controlo através de credenciais utilizando *Basic Authentication*. Ao definir credenciais inválidas para o pedido é formada uma *string* <nome de utilizador>:<palavra-passe> que, após codificada na variante RFC2045-MIME de Base64 e concatenada com o excerto "Basic ", é enviado no cabeçalho responsável pela transferência de informações de autenticação como apresentado na secção 1 da figura 4.2.

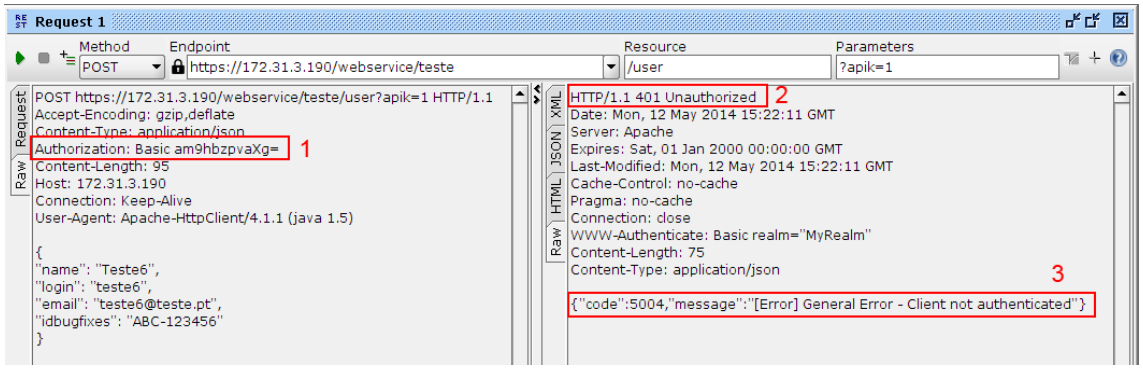

Figura 4.2: Teste de validação do controlo de autenticação

Comparando o valor enviado com o expectável, neste caso a string "Basic am9hbzpvaQ==", a API infere que o pedido não possui autorização para aceder às suas funcionalidades enviando como resposta o código HTTP "*401 Unauthorized*"e no seu conteúdo o código de erro correspondente e uma mensagem descritiva. Apesar de desativada durante todo o processo de desenvolvimento e execução de testes através de uma máquina remota, a solução desenvolvida garante a rejeição de pedidos não locais como é possível visualizar na figura 4.3 após executado um pedido com a funcionalidade ativa.

De seguida procedeu-se ao testes aos serviços fornecidos pela API. Tendo em consideração a divisão das funcionalidades dos serviços pelos métodos HTTP utilizados é possível assumir que as respostas a cada método sejam análogas e consistentes ao longo dos serviços pelo que um pedido GET de um recurso específico terá sempre como resposta os detalhes acerca desse recurso, quer seja uma regra de *firewall*, quer

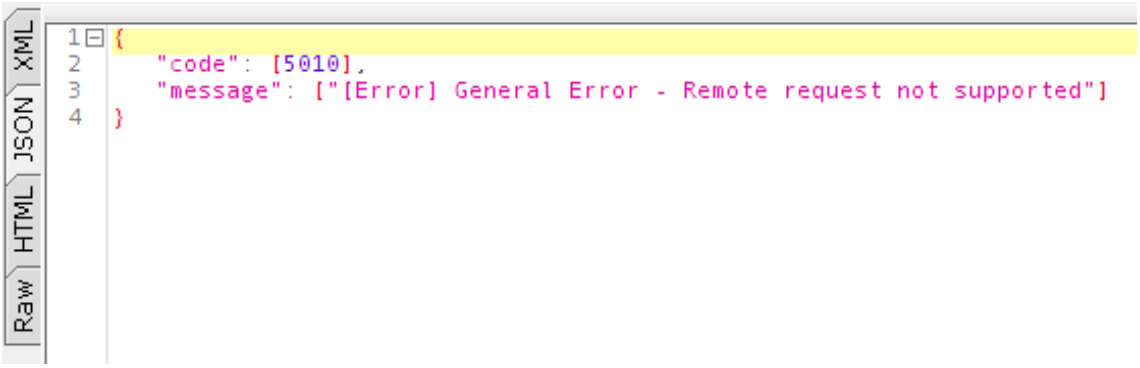

Figura 4.3: Teste de validação do controlo de origem do pedido

seja um registo de DNS. Isto permite-nos avaliar as respostas aos pedidos de teste de maneira mais eficaz procurando sempre o padrão correspondente à resposta pretendida. De maneira a manter o documento sucinto e não redundante, serão omitidas figuras e resultados de todas funcionalidades de todos os serviços, sendo apenas apresentados os diferentes padrões de respostas possíveis. No entanto é importante salientar que no decorrer do projeto este tipo de testes foi executado para todas as funcionalidades oferecidas pela solução.

Pedidos GET

Um pedido GET serve como um método de obtenção de informação, sem intencionalidade de modificação do sistema. Ao ser executado sobre um URI genérico de um serviço ou sub-serviço, obtém como resposta uma lista de recursos desse tipo. No exemplo é efetuado um pedido ao serviço *firewall* e, sendo que o recurso passível de ser gerido através deste serviço é a regra de *firewall*, é obtida uma lista com todas as regras registadas no sistema sendo que, dependendo da implementação, poderá apresentar detalhes completos dos recursos ou apenas elementos identificativos.

É também possível efetuar pedidos GET a certos serviços com a especificação de valores de filtragem da lista a ser retornada, limitando assim o número de representações de recursos na resposta. Embora não seja proibitiva a utilização de uma mensagem no corpo de um pedido *GET*, optou-se por estruturar a API desenvolvida de maneira a não considerar essa secção do pedido e apenas obter os parâmetros através do URL.

Por fim, quando se trata de pedidos a URIs identificadores de um recurso, a resposta obrigatoriamente, e apenas caso exista, retornará a representação desse recurso com todos os seus detalhes.

No caso da não existência do recurso indicado pelo URI ao qual foi executado o pedido, é retornada essa informação, quer por meio do código acHTTP *"404 Not Found"* quer por uma mensagem na resposta mais específica do serviço em questão (figura 4.7).

Pedidos POST

Simulando um contexto de criação de um recurso, é possível visualizar que a um pedido incorretamente formulado (figura 4.8) a API não executa a ação desejada e responde com informação relevante sobre o erro de formulação do pedido, tanto no corpo da mensagem (figura 4.9) como no código HTTP da resposta (figura 4.10).

Ao corrigir o pedido (figura 4.13), tratando-se da criação de um recurso no sistema, é retornado no corpo da resposta os detalhes do recurso criado (figura 4.12) como se de um GET se tratasse. Desta maneira é possível do lado do cliente, sem necessidade de efetuar mais nenhum pedido, validar o sucesso do pedido.

Figura 4.4: Pedido de listagem de regras de *firewall* e resposta

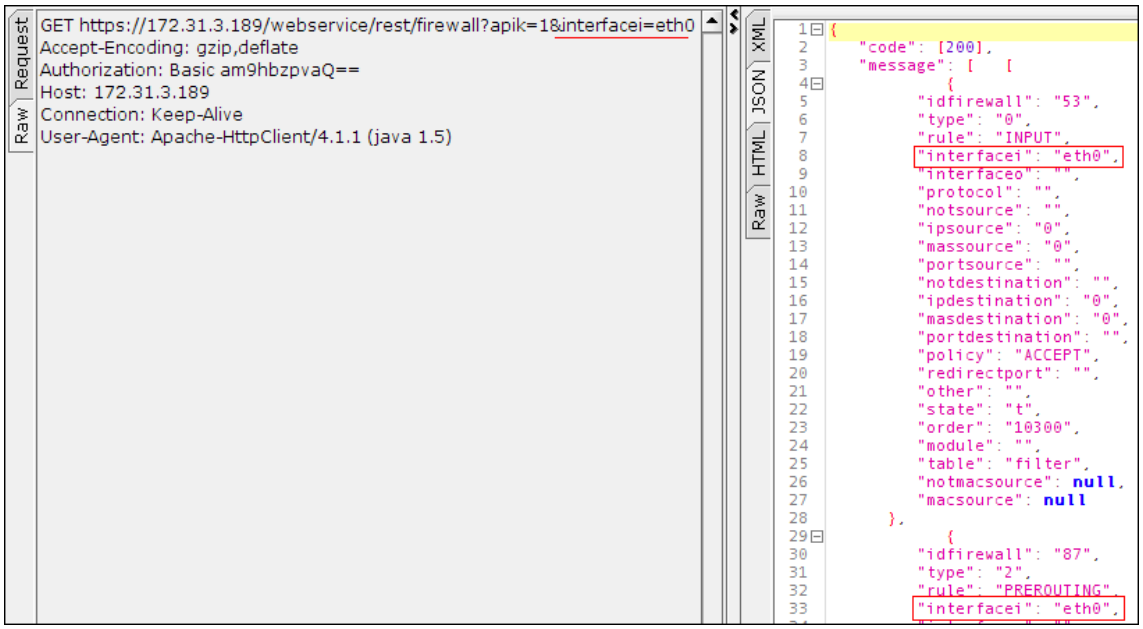

Figura 4.5: Pedido de listagem de regras de *firewall* com definição de filtro e resposta

Adicionalmente, o código HTTP da resposta também retrata a maneira como foi conduzido o pedido, indicando a ocorrência da criação de um recurso no sistema (figura 4.13).

Pedidos PUT

Um pedido do tipo PUT oferece um método de modificação de um recurso definido por um URI ao qual o pedido PUT é efetuado. Caso exista, o pedido é processado à semelhança de um pedido POST, sendo anali-

Figura 4.6: Pedido de detalhes de regra de firewall e resposta

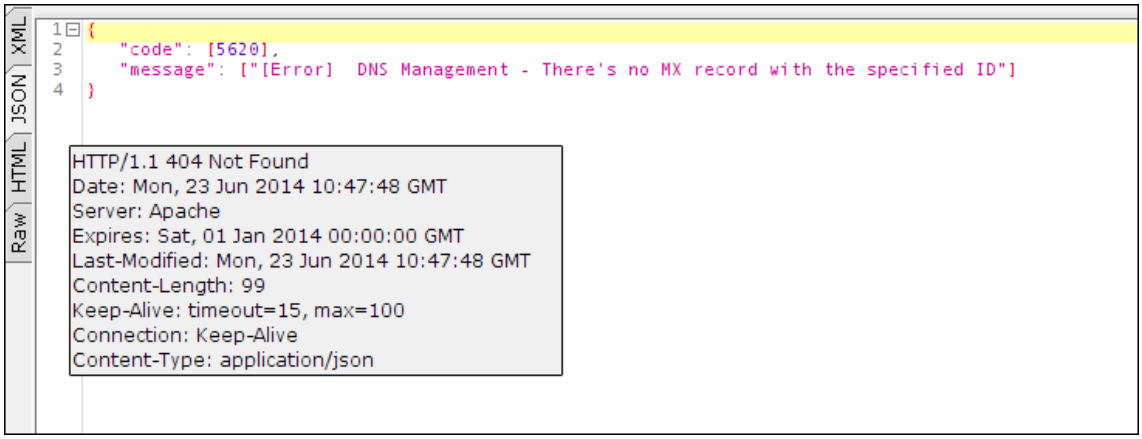

Figura 4.7: Resposta a pedido mal-sucedido

sados os parâmetros enviados no corpo do pedido e, na eventualidade de ser bem-sucedido, será retornada a nova representação do recurso modificado para, mais uma vez, uma mais eficaz e rápida validação da ação desejada como apresentado na figura 4.14. Caso contrário, é retornada uma resposta com o código HTTP *"404 Not Found"*.

Pedidos DELETE

Pedidos do tipo DELETE são utilizados para eliminação de um recurso existente no sistema. No exemplo seguinte é demonstrada a eliminação de um registo MX e validada esta ação recorrendo às próprias funcionalidades da API. Ao efetuar um pedido de GET para o serviço gestor de registos MX é obtida a lista de todos os recursos deste tipo registados no sistema.

Após verificar a existência do recurso a eliminar é então enviado o pedido DELETE para o URI identificador desse recurso. Neste caso, pretende-se eliminar o registo MX com o identificador 2, razão pela qual é enviado o pedido para o URI /dns/mx/2 (figura 4.16).

```
POST https://172.31.3.189/webservice/rest/firewall?apik=1
Accept-Encoding: gzip,deflate
Content-Type: application/json
Authorization: Basic am9hbzpvaQ==
Content-Length: 89
Host: 172.31.3.189
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
"rule": "FORWARD",
"policy": "drop it",
"table": "filter",
"idbugfixes": "ABC-123457"
}
```

Figura 4.8: Pedido de criação de regra de *firewall* com parâmetros errados

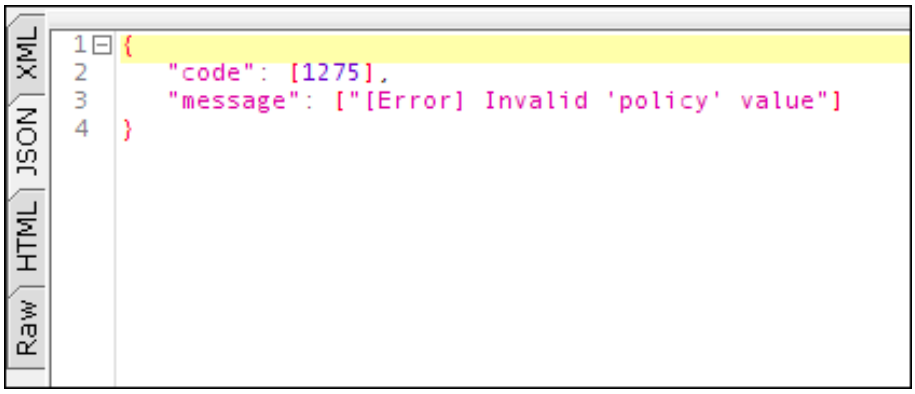

Figura 4.9: Resposta de criação de regra de *firewall* com parâmetros errados

```
HTTP/1.1 400 Bad Request
Date: Mon, 23 Jun 2014 10:50:48 GMT
Server: Apache
Expires: Sat, 01 Jan 2014 00:00:00 GMT
Last-Modified: Mon, 23 Jun 2014 10:50:48 GMT
Content-Length: 61
Connection: close
Content-Type: application/json
```

Figura 4.10: Cabeçalho da resposta à criação de regra de *firewall* com parâmetros errados

É obtida então uma resposta indicando o sucesso da eliminação do recurso (figuras 4.17)

E, por fim, é possível validar todo o processo mais uma vez recorrendo à lista de registos MX, desta vez sem o registo previamente eliminado (figura 4.18).

```
POST https://172.31.3.189/webservice/rest/firewall?apik=1
Accept-Encoding: gzip,deflate
Content-Type: application/json
Authorization: Basic am9hbzpvaQ==
Content-Length: 86
Host: 172.31.3.189
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
"rule": "FORWARD",
"policy": "DROP",
"table": "filter",
"idbugfixes": "ABC-123457"
}
```

Figura 4.11: Pedido de criação de regra de *firewall* com parâmetros corretos

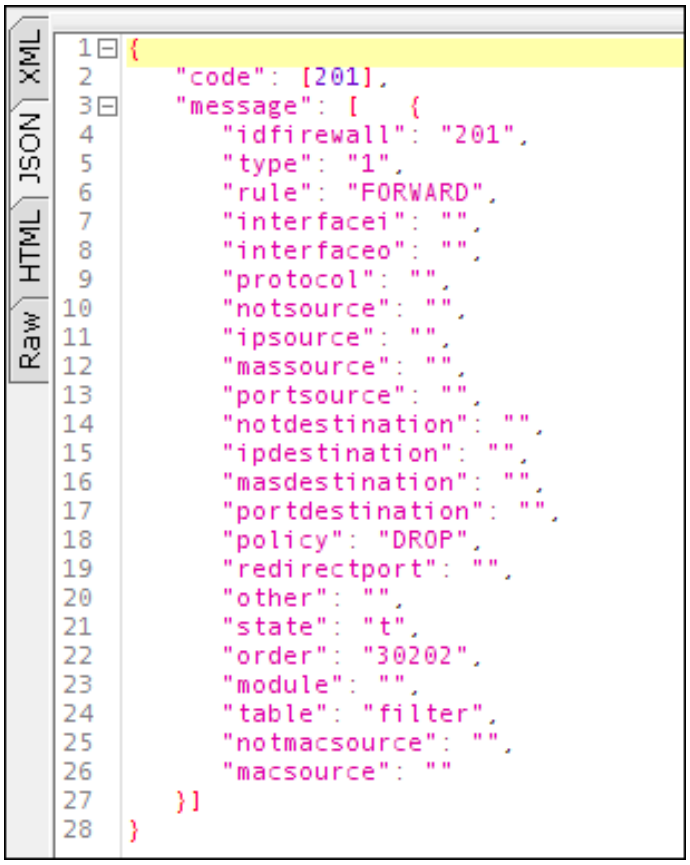

Figura 4.12: Resposta de criação de regra de *firewall* com parâmetros corretos

4.3 Testes de Desempenho

4.3.1 Estrutura dos Testes

De maneira a garantir uma simulação coerente da utilização da API num contexto de instalação de um aplicação num sistema IPBrick, foi organizado um caso de teste formado por pedidos de criação e manipu-

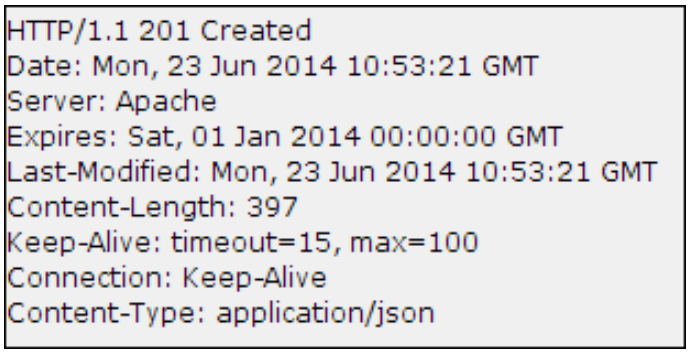

Figura 4.13: Cabeçalho da resposta à criação de regra de *firewall* com parâmetros errados

Figura 4.14: Pedido de modificação de conta de utilizador e resposta

Figura 4.15: Lista de registos MX prior à eliminação

lação de um *virtual host* com parâmetros gerados aleatoriamente por execução. A garantia de exclusividade no acesso aos recursos do sistema é fornecida através de um pedido inicial para começo do processo de instalação. No caso de outra instalação já se encontrar a decorrer, o pedido é rejeitado e o cliente obtém a informação de que qualquer pedido posterior poderá incorrer num erro de concorrência e afetar o sistema. A libertação dos recursos ocorre aquando o pedido para finalização de uma instalação, altura em que qualquer outro processo poderá requisitar acesso exclusivo na instalação de uma aplicação. Utilizando a funcionalidade incorporada na ferramenta SOAP UI de execução de testes de sobrecarga de um serviço foi possível

```
DELETE https://172.31.3.189/webservice/rest/dns/mx/2?apik=1
Accept-Encoding: gzip,deflate
Authorization: Basic am9hbzpvaQ==
Host: 172.31.3.189
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

Figura 4.16: Pedido de eliminação de registo MX

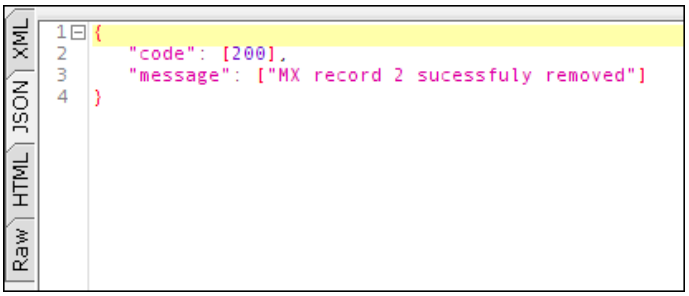

Figura 4.17: Resposta ao pedido de eliminação do registo MX

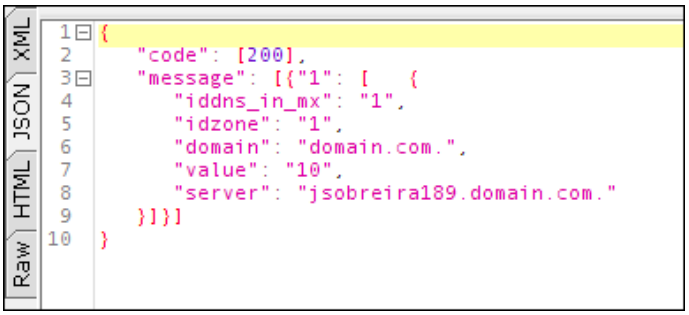

Figura 4.18: Lista de registos MX após eliminação

simular mais do que um processo de instalação simultâneo e diferentes intervalos de tempo de espaçamento entre processos. Para uma simulação coerente foi necessário adicionar algumas condicionantes. Tendo em conta que todos os casos de teste teriam de ser iniciados com um pedido POST /inst e terminados com um pedido DELETE /inst mesmo que algum pedido intermédio falhasse foi indispensável garantir que o pedido de finalização de instalação seria sempre executado, caso contrário todos os processos de instalação seguintes iriam falhar por existência de uma instalação a decorrer. A solução passou por correr um caso de teste de acesso aos serviços (figura 4.19) dentro do caso de teste a executar (figura 4.20). Seria ainda necessário definir que o caso de teste principal seria identificado como sendo mal-sucedido se o de acesso aos serviços falhasse, sendo este último cancelado e o principal continuado até ao último pedido. O teste ao serviço consistiu na criação de um *virtual host*, obtenção dos dados acerca do mesmo e de uma lista de todos os *virtual hosts* do sistema e eliminação do *virtual host* criado. Era assim garantido que, ao longo da execução dos processos, todos os recursos criados seriam eliminados evitando erros por manipulação de recursos criados por outros processos. De maneira a conseguir a obtenção do identificador do recurso criado e consequente utilização pelos pedidos seguintes foi elaborado um *script* Groovy capaz de extrair esse valor da resposta do primeiro pedido, registar o valor numa variável local possível de ser acedida por qualquer outro pedido (4.3.1).

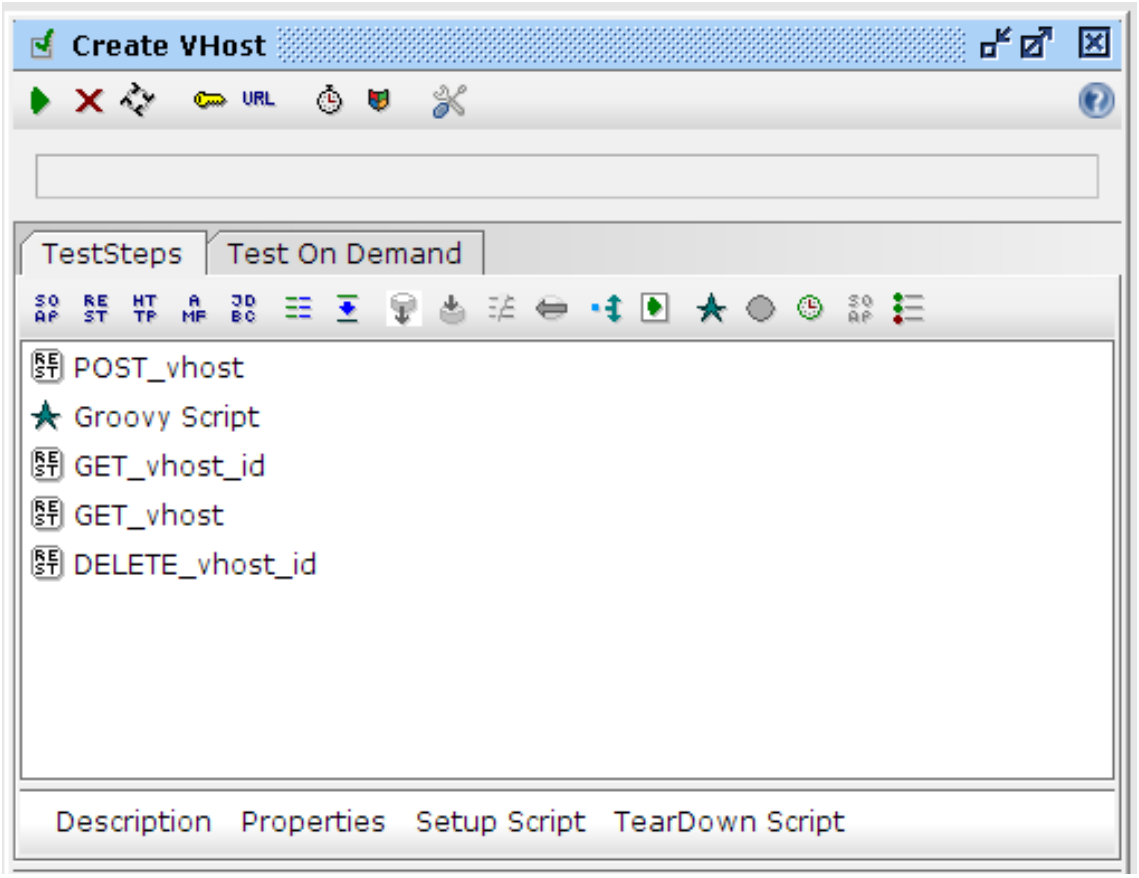

Figura 4.19: Caso de teste de acesso ao serviço de gestão de *virtual hosts*

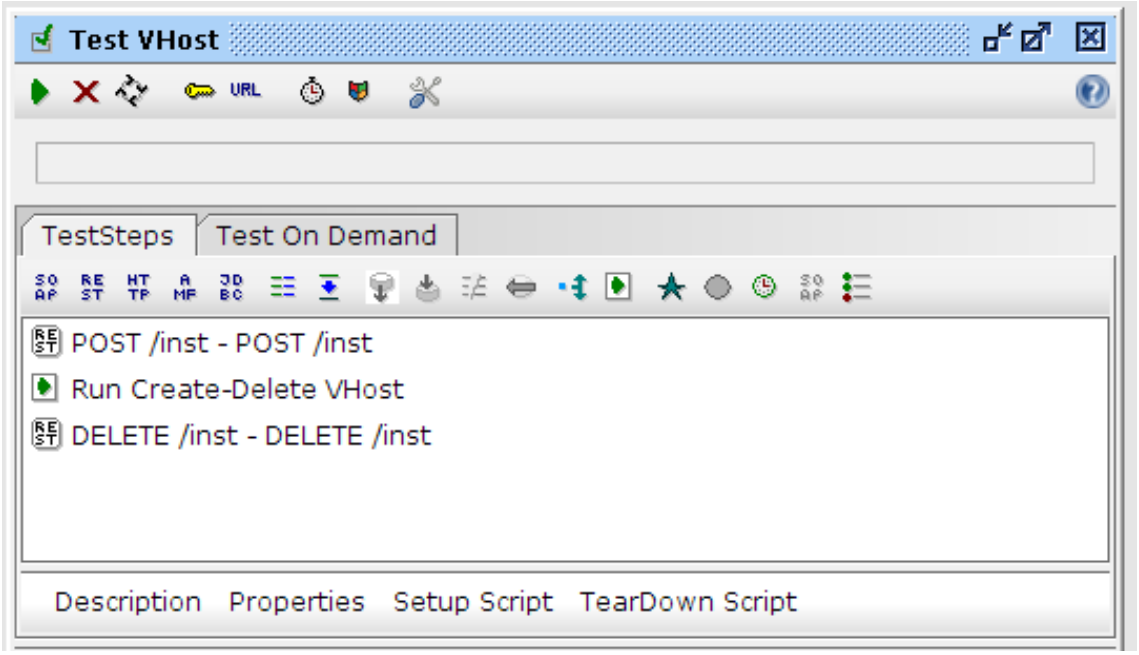

Figura 4.20: Caso de teste principal

```
1  import groovy.json.JsonSlurper
2
3  responseContent = testRunner.testCase.getTestStepByName("POST_vhost
       ").getPropertyValue("response")
4  slurperresponse = new JsonSlurper().parseText(responseContent)
5  def code = slurperresponse.code[0]
6  def idapache = slurperresponse.message[0].idapache
7  log.info("idapache: "+idapache)
8  log.info("code: "+code)
9  targetStep1 = testRunner.testCase.getTestStepByName("GET_vhost_id")
10 targetStep1.setPropertyValue( "idvhost", idapache)
11 targetStep2 = testRunner.testCase.getTestStepByName("
       DELETE_vhost_id")
12 targetStep2.setPropertyValue( "idvhost", idapache)
```

Listing 4.1: *Script* Groovy de extração e atribuição do identificador de recurso

4.3.2 Resultados

O objetivo principal deste tipo de teste visava uma análise simples da variação da taxa de erros e do tempo de execução do processo de instalação nos diferentes contextos referidos previamente. Foi possível implementar mais do que um processo a correr simultaneamente com recurso à parametrização de um número de *threads* desejado na ferramenta *LoadTest* do programa SOAP UI. Desta forma foi possível avaliar o comportamento da solução numa situação de sobrecarga de pedidos de inicio de instalação. É importante referir que uma autorização não concedida a um pedido deste género resulta na identificação de um erro no caso de teste e todos os casos de teste foram parametrizados com um *delay* médio de 1000 ms.
Executando o teste com uma, duas e quatro *threads* simultâneas foi possível obter uma percentagem de erro por caso de teste apresentado na figura 4.21. Como seria expectável apenas um processo de instalação a decorrer não produz nenhum erro pois não ocorre um problema de concorrência. No entanto, ao aumentar para dois processos de instalação simultâneos a percentagem de erros sobe dramaticamente para um valor de aproximadamente um erro por cada dois casos de teste. Ao aumentar o número de *threads* para o dobro é possível verificar um aumento de 51% na percentagem de erro, atingindo o valor de 74% ou cerca de 3 erros por cada 4 casos de teste. Estes resultados vêm confirmar a tendência para apenas um processo garantir o acesso aos recursos, obrigando os outros a esperar pela sua finalização. Um fator também importante para este teste é a distribuição dos erros ocorridos. Analisando os resultados obtidos previamente e o funcionamento habitual da aplicação é possível antecipar uma maior frequência de erros nos pedidos de início de instalação.
Ao analisar os valores obtidos neste campo confirma-se o previsto com uma clara maioria dos erros a serem identificados no pedido POST /inst e erros ocasionais a ocorrerem noutras alturas do processo. Outro fator que exerce influência na percentagem de erros obtidos refere-se ao intervalo de tempo entre inícios de processos de instalação (figura 4.23).
A execução de pedidos de inicio de instalação mais espaçados permite obter um menor número de rejeições tomando em consideração o tempo necessário para um processo concluir a instalação da aplicação.

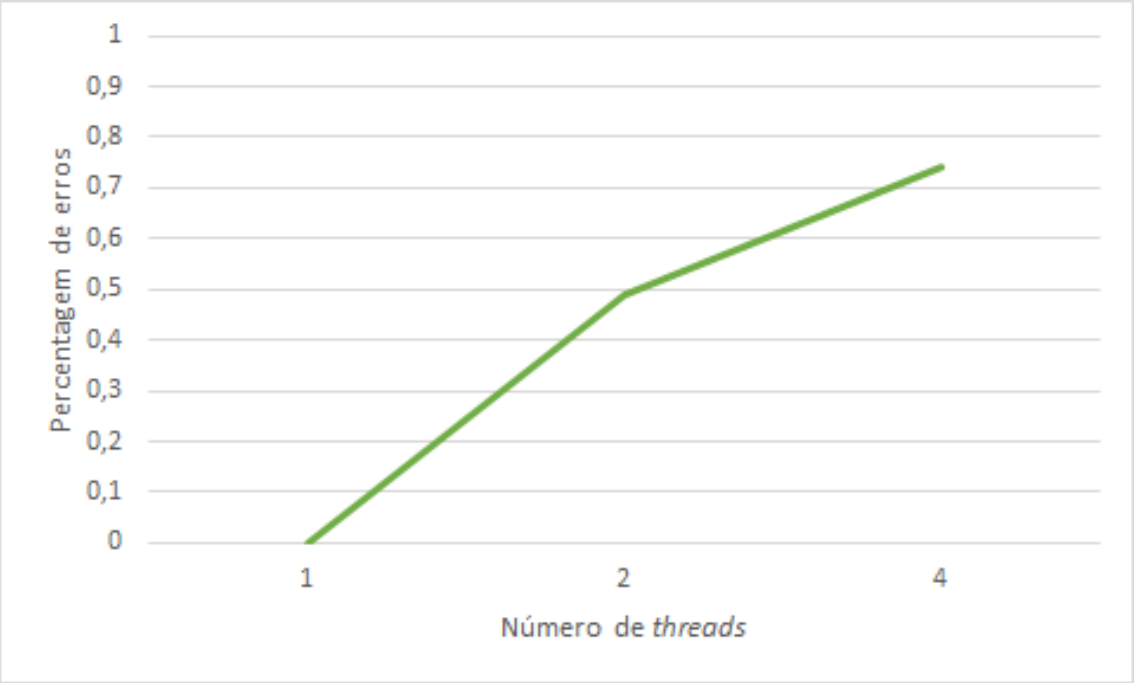

Figura 4.21: Evolução da percentagem de erro em casos de teste em relação ao número de *threads* total

Figura 4.22: Distribuição de erros por etapas em 2 e 4 *threads* em execução

4.4 Testes de Segurança

4.4.1 Estrutura dos Testes

Para verificação da segurança da API foram também utilizados os testes incluídos no programa SOAP UI, sendo que este apresenta uma vasta gama de valores predefinidos para identificação de vulnerabilidades em parâmetros enviados. A API foi testada para Cross Site Scripting (XSS), *fuzzing scan*, tipos inválidos de parâmetros e injeção SQL e XPath sendo utilizado o serviço de gestão de *virtual hosts* e todos os parâmetros costumizáveis num acesso a este serviço, nomeadamente valores embutidos no URL, como é o caso da chave API e dos identificadores, credenciais de autenticação e, caso seja válida a sua inclusão, conteúdo do corpo do pedido. Para validação automática dos resultados foram adicionadas condições ao código HTTP

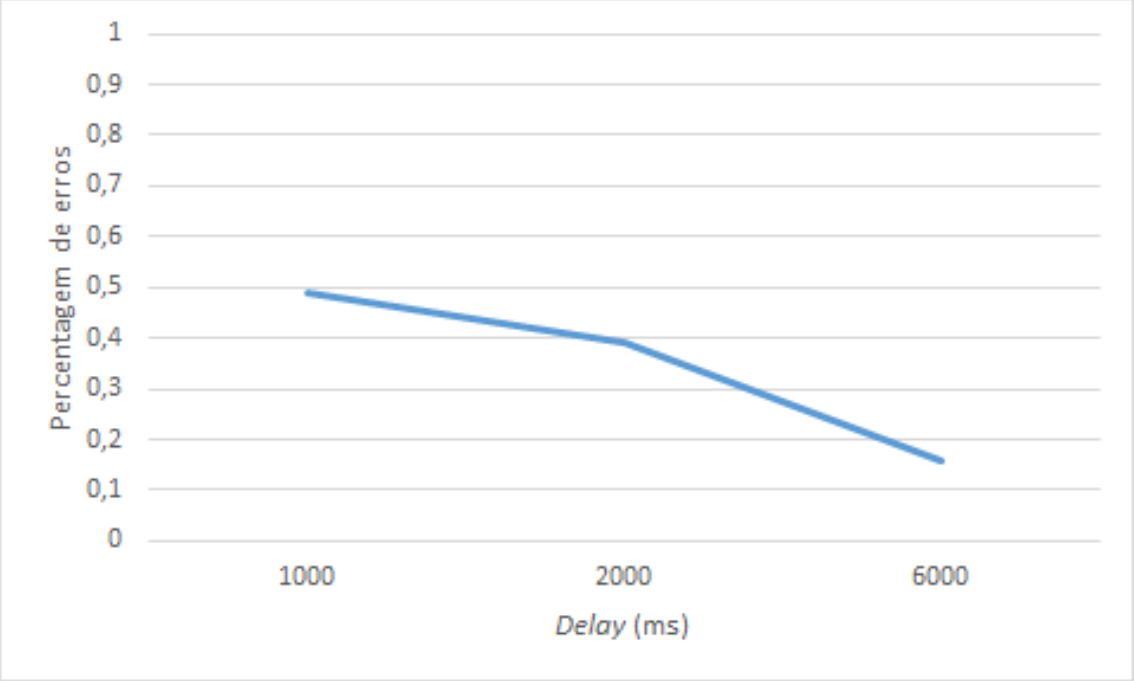

Figura 4.23: Evolução da percentagem de erro em casos de teste em relação ao *delay* definido

de cada resposta, sendo que um código 200 ou 201 indicaria a não deteção da existência de um ou mais parâmetros inválidos e existência de uma potencial vulnerabilidade.

4.4.2 Resultados

GET /vhost

Como referido em 4.4.1 a ferramenta de validação dos teste de segurança necessitam a identificação de um ou mais pontos de entrada de valores introduzidos pelo cliente, pontos esses suscetíveis de serem utilizados na tentativa de aproveitar falhas da solução. No caso de um pedido GET, esses parâmetros são reduzidos aos comuns a todos os tipos de pedidos por exclusão do corpo da mensagem, sendo estes a chave da API e URI (incluídos no URL do pedido) e as credenciais de acesso (incluídas no cabeçalho do pedido *Authorization*). Durante a execução dos testes à funcionalidade de obtenção de detalhes de *virtual hosts* foram identificadas quatro potenciais vulnerabilidades sendo duas resultado do teste a XSS e duas resultado do teste aos tipos de parâmetros inválidos.

Analisando o registo dos testes é possível verificar a não existência de fugas de informação relevantes, sendo os alertas dos testes de XSS resultado quer de uma resposta de erro com o parâmetro enviado a ser retornado, quer por um corpo de resposta vazio por força do mecanismo de segurança da API.

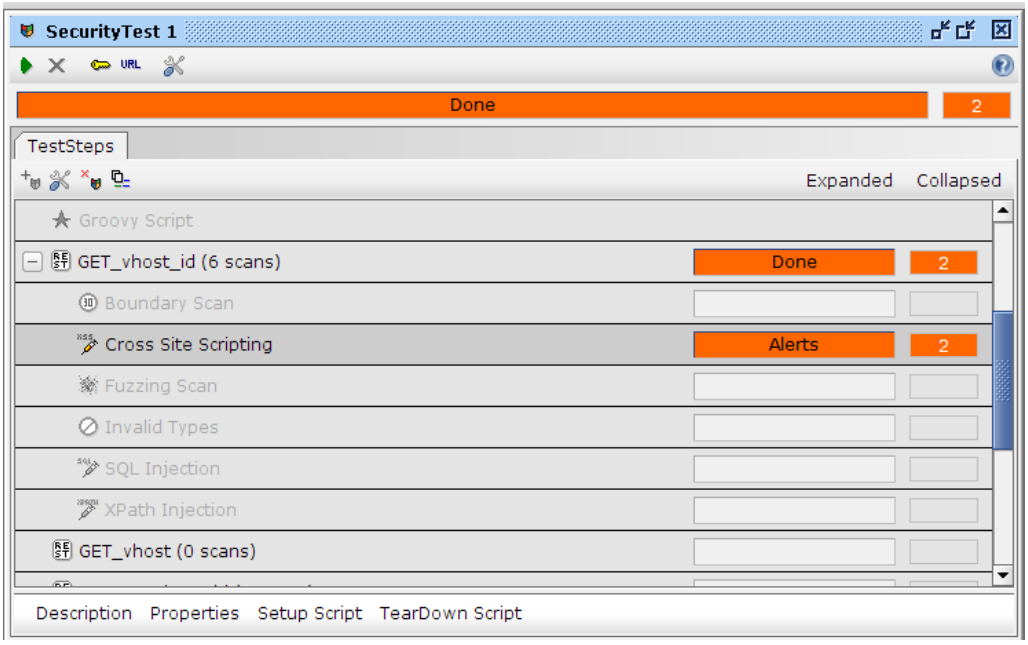

Figura 4.24: Resultados do teste de segurança a pedidos GET - *Cross Site Scripting*

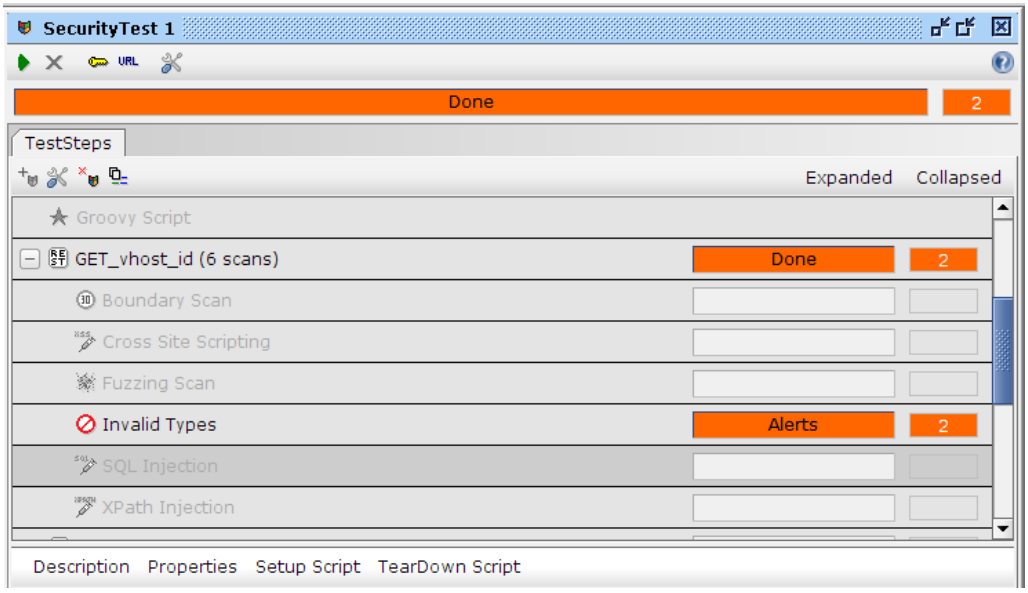

Figura 4.25: Resultados do teste de segurança a pedidos GET - Parâmetros de tipo inválido

```
1 [Cross Site Scripting] Request 42 - FAILED - [idvhost=<script>alert
    ('XSS')</script>]: took 5019 ms
2  -> Content that is sent in request '<script>alert('XSS')</...␣' is
     exposed in response. Possibility for XSS script attack in:
    GET_vhost_id
3 [Cross Site Scripting] Request 70 - FAILED - [idvhost=<BODY onload
    !#$%&()*~+-_.,:;?@[/|\]^=alert("XSS")>]: took 14 ms-> null/empty response->
    Status code extraction error!
```

Listing 4.2: *Log* do teste de segurança a pedidos GET - *Cross Site Scripting*

Já no caso dos testes de tipos de parâmetros inválidos, a utilização do valor 1 como parâmetro de teste voltou a provocar um alerta pela existência de um recurso com um identificador de valor 1. Numa análise mais profunda ao segundo alerta foi possível verificar que, apesar de não ser apresentada nenhuma mensagem de erro pela API, o sistema de gestão de base de dados informa no seu registo local acerca da inválida atribuição de um valor superior à gama aceite pelo atributo definido como inteiro.

```
1  [Invalid Types] Request 22 - FAILED - [idvhost=1]: took 142 ms
2  -> Response status code: 200 is in invalid list of status codes
3  [Invalid Types] Request 24 - FAILED - [idvhost
       =882223334991111111]: took 243 ms
4  -> Response status code: 200 is in invalid list of status codes
```

Listing 4.3: *Log* do teste de segurança a pedidos GET - Parâmetros de tipo inválido

POST /vhost

Com a necessidade de introdução de um corpo de mensagem num pedido POST é nos apresentado outro ponto de possível vulnerabilidade da API. É assim necessário garantir a inclusão deste parâmetro na gama de testes a executar. Não foram identificadas potenciais falhas nos testes de XSS, *fuzzing scan*, injeção SQL e injeção XPath, sendo apenas apresentado um alerta no teste de parâmetros de tipo inválido.

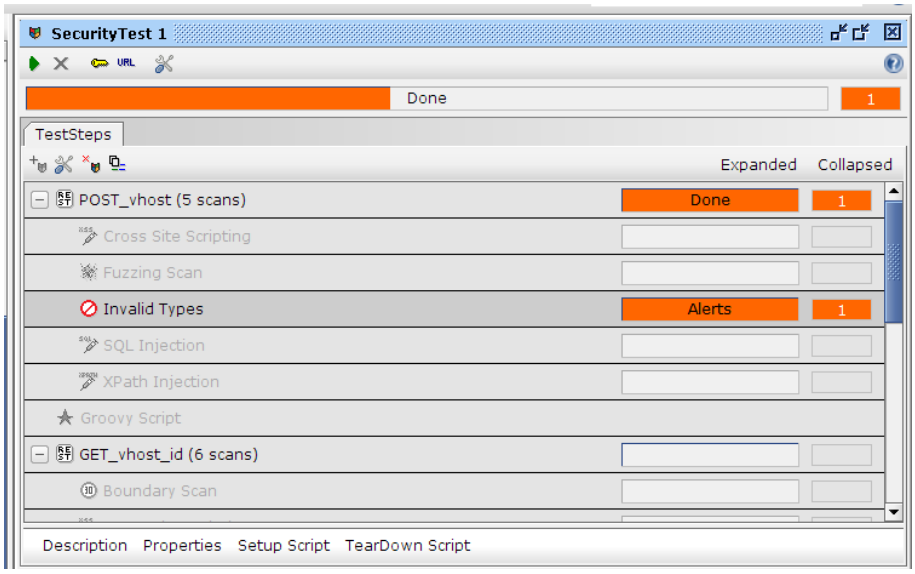

Figura 4.26: Resultados do teste de segurança a pedidos POST - Parâmetros de tipo inválido

Recorrendo ao registo do teste, é possível verificar que o alerta é apresentado pelo fato da chave API ter sido, involuntariamente, descoberta. Para efeitos de teste, a escolha da chave API recaiu sobre um valor simples e apenas de demonstração, pelo que a utilização de um valor mais robusto facilmente evitaria esta situação.

```
1 [Invalid Types] Request 97 - FAILED - [apik=1]: took 209 ms
```

Listing 4.4: *Log* do teste de segurança a pedidos POST - Parâmetros de tipo inválido

PUT /vhost e DELETE /vhost
Mais uma situação em que se deve garantir a cobertura do corpo da mensagem no espectro de análise dos testes de segurança no caso do pedido do tipo PUT. No entanto, os resultados de ambas as baterias de teste apresentam os mesmos indicadores apresentados até ao momento por utilização do parâmetro 'idvhost' e 'apik' no URL dos pedidos.

4.5 Testes de Integração

De maneira a exemplificar o funcionamento da solução apresentada foram organizados testes de integração de aplicações em pacotes adaptados. Numa primeira fase foi criado um pacote-exemplo, construído desde o inicio com o intuito de abordar todas as funcionalidades da API. Mais sobre este pacote é descrito em 4.5.1 desde a sua elaboração até à execução da sua instalação no sistema IPBrick disponível. Posteriormente a disponibilização de duas aplicações IPBrick permitiu a execução de testes mais complexos, desde a adaptação de um pacote já final até à identificação de possíveis incompatibilidades, servindo como prova de validação final de implementação da solução num produto de acesso público.

4.5.1 Aplicação de Teste

Sendo desenvolvida de raiz, o objetivo desta aplicação seria o de garantir o acesso ao máximo número de funcionalidades da API mantendo o maior grau de simplicidade possível.

4.5.1.1 Estrutura dos Testes

O pacote da aplicação foi então projetado como contendo além do obrigatório ficheiro de controlo, também os quatro possíveis ficheiros de execução de *scripts* em estados diferentes da instalação ou desinstalação do mesmo.

```
jsobreira@posto51:~/Documentos/PACOTES/TESTE/pacote_cliente$ cat DEBIAN/control
Package: apiclient
Version: 0.2
Maintainer: João Sobreira <jsobreira@est.ipbrick.com>
Architecture: all
Description: Client for IPBrick API
```

Figura 4.27: Conteúdo do ficheiro control do pacote de teste

Para manutenção da simplicidade do código dos ficheiros constituintes, foi desde logo definida uma divisão dos mesmos em três partes claramente definidas:

1. Definição de funções
2. Definição de variáveis
3. Execução de pedidos

```
jsobreira@posto51:~/Documentos/PACOTES/TESTE/pacote_cliente$ ls -Bla DEBIAN
total 63
drwxr-xr-x 2 nobody nogroup  1024 Abr 14 16:22 .
drwxr-xr-x 4 nobody nogroup  1024 Abr  4 16:29 ..
-r-xr-xr-x 1 nobody nogroup   272 Abr 14 16:09 control
-rwxr-xr-x 1 nobody nogroup 11514 Abr 14 16:21 postinst
-rwxr-xr-x 1 nobody nogroup  9551 Abr 14 16:22 postrm
-rwxr-xr-x 1 nobody nogroup 20498 Abr 14 16:22 preinst
-rwxr-xr-x 1 nobody nogroup 15187 Abr 14 16:22 prerm
```

Figura 4.28: Estrutura do pacote de teste

Desta maneira foi possível a reutilização de código ao longo de cada ficheiro e fácil manutenção e alteração do mesmo, garantindo coerência no conteúdo dos diferentes dos pedidos e evidenciando a universalidade e simplicidade de acesso à API de integração. Na secção de definição de funções, destaque para a função principal responsável pela elaboração de um pedido HTTP com base nos parâmetros fornecidos.

```
1  function send_request($url, $method, $headers, $auth, $data, &
       $response, &$code){
2
3          $handle = curl_init();
4          curl_setopt($handle, CURLOPT_URL, $url);
5          curl_setopt($handle, CURLOPT_HTTPHEADER, $headers);
6          curl_setopt($handle, CURLOPT_USERPWD, $auth["username"] . "
               :" . $auth["password"]);
7          if($method=='POST'){
8                  $curl_data = json_encode($data);
9                  curl_setopt($handle, CURLOPT_POST, 1);
10                 curl_setopt($handle, CURLOPT_POSTFIELDS, $curl_data
                       );
11         }
12         if($method=='PUT'){
13                 $curl_data = json_encode($data);
14                 curl_setopt($handle, CURLOPT_CUSTOMREQUEST, "PUT");
15                 curl_setopt($handle, CURLOPT_POSTFIELDS, $curl_data
                       );
16         }
17         if($method=='DELETE'){
18                 curl_setopt($handle, CURLOPT_CUSTOMREQUEST, "DELETE
                       ");
19         }
20         if($method=='OPTIONS'){
21                 curl_setopt($handle, CURLOPT_CUSTOMREQUEST, "
                       OPTIONS");
22         }
23         curl_setopt($handle, CURLOPT_RETURNTRANSFER, true);
24         curl_setopt($handle, CURLOPT_SSL_VERIFYHOST, false);
25         curl_setopt($handle, CURLOPT_SSL_VERIFYPEER, false);
```

```
26          $response = curl_exec($handle);
27          $code = curl_getinfo($handle, CURLINFO_HTTP_CODE);
28          curl_close($handle);
29  }
```

Listing 4.5: Função de envio de pedido

Utilizando a biblioteca *cURL* nativamente incorporada na versão atual PHP, a função recebe como parâmetros o URL destino do pedido, o método a utilizar, cabeçalhos adicionais, credenciais de autenticação e, caso seja necessário, conteúdo a enviar no corpo do pedido. É também passado à função dois apontadores para variáveis onde serão guardados quer o conteúdo da resposta, quer o seu código HTTP. Apesar de ser de elaboração simples é importante identificar um ponto essencial na linha 8 e repetido na linha 13 da função apresentada, onde o vetor passado à função contendo informação adicional é convertido em formato JSON para ser então embutido no pedido final e, mais tarde, convertido novamente num vetor para mais fácil manipulação. Sendo o conteúdo da secção de definição de variáveis apenas a atribuição de valores predefinidos e específicos da aplicação a variáveis enviadas diretamente ou indiretamente à função acima apresentada passamos à análise da execução dos pedidos.

```
1
2  // Send request to start installation
3  // POST /inst
4
5  $service = "inst";
6  $method = 'POST';
7  $url = $base_url.'/'.$service.'?apik='.$apik; // URL TO SEND
       REQUEST
8
9  $post_data = array( // POST INFORMATION
10                 "version" => $version,
11                 "idbugfixes" => $idbugfixes,
12                 "dependencies" => $dependencies,
13                 "conflicts" => $conflicts
14 );
15
16 send_request($url, $method, $headers, $auth, $post_data, $response,
       $code);
17 $response_post_inst = json_decode($response,true);
18 $code_post_inst = $code;
19
20 echo $method." /".strtoupper($service)." <br /> \n";
21 echo "HTTP Code: ".$code_post_inst."<br /> \n";
22 echo "Response: ";
23 echo "<pre>";
24 var_dump($response_post_inst);
25 echo "</pre>";
```

```
26  echo "<br /><br />\n\n";
27
28  if($response_post_inst["code"]!=200 || $code_post_inst!=200){
29          exit(1);
30  }
```

Listing 4.6: Exemplo de chamada à função de envio de pedido

Em ordem a homogenizar as chamadas à função responsável pela execução dos pedidos, foi mantida a definição dos parâmetros específicos de cada pedido prior a cada chamada. Com base nestes elementos estrutura-se então as ações dos ficheiros do pacote na seguinte maneira:

preinst

1. Inicio da instalação (POST /inst)
2. Criação de um utilizador (POST /user)
3. Criação de um *virtual host* (POST /vhost)
4. Criação de uma regra na firewall do sistema (POST /firewall)
5. Criação de uma zona DNS (POST /dns/zone)
6. Criação de dois registos DNS do tipo A (POST /dns/a)
7. Criação de um registo DNS do tipo NS (POST /dns/mx)
8. Criação de um registo DNS do tipo MX (POST /dns/mx)
9. Criação de um registo DNS do tipo SRV (POST /dns/srv

postinst

1. Registo da aplicação na lista de *bugfixes* (POST /bugfix)
2. Finalização da instalação (DELETE /inst)

prerm

1. Inicio da instalação (POST /inst)
2. Eliminação do utilizador criado (DELETE /user/<iduser>)
3. Eliminação do *virtual host* criado (DELETE /vhost/<idvhost>)
4. Eliminação da regra de *firewall* criada (DELETE /firewall/<idfirewall>)
5. Eliminação da zona criada (DELETE /dns/zone/<idzone>)

postrm

1. Eliminação do registo da aplicação da lista de *bugfixes* (DELETE /bugfix/<idbugfixes>)
2. Finalização da instalação (DELETE /inst)

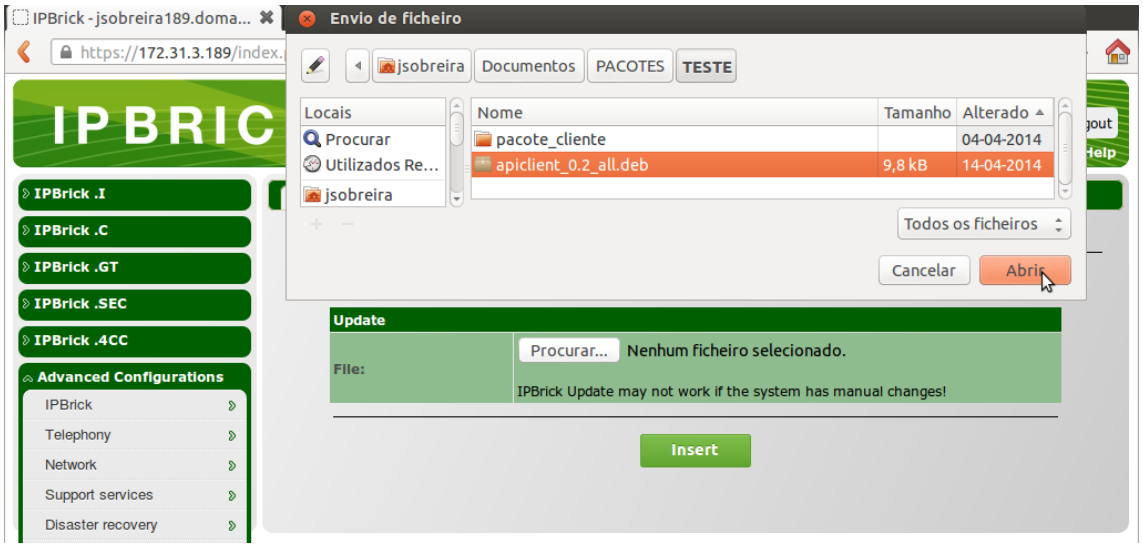

Figura 4.29: Instalação do pacote no sistema IPBrick

4.5.1.2 Resultados

Instalação

Compilando então o código num pacote *debian* procedeu-se à sua instalação pela interface IPBrick.

O *output* da instalação foi analisado e, não tendo sido identificados erros, foi possível efetuar a confirmação da correta instalação da aplicação.

Seguindo a ordem da instalação, na secção da lista de contas registadas no sistema IPBrick foi possível confirmar a existência do utilizador especificado (figura 4.30).

Name	Login	Account	Expires in
Administrator	administrator	Local	Not defined
username	userlogin	Local	Not defined

Figura 4.30: Conta de utilizador criada pela aplicação-teste

De seguida, já na secção de *Web Servers* também foi identificado o *virtual host* criado pela aplicação aquando a sua instalação. (figura 4.31).

De maneira a garantir a correta adição da regra especificada na firewall do sistema, procedeu-se ao acesso através da linha de comando e listagem de todas as regras registadas, tendo sido identificada uma única correspondente às caraterísticas pretendidas (figura 4.32).

De volta à interface gráfica do sistema IPBrick, na secção correspondente a configurações do servidor DNS confirmou-se a criação de uma nova zona DNS (figura 4.33) e, relativa a esta, os registos criados (a figura 4.34 apresenta os registos do tipo A criados).

Por fim também foi possível verificar o registo da instalação da aplicação na lista de *bugfixes* com os detalhes definidos no pacote (figura 4.35).

É assim validada a instalação da aplicação de teste com a utilização da solução desenvolvida.

Desinstalação

Dado o fato do sistema IPBrick não estar configurado para permitir a desinstalação de uma aplicação através

Web site definitions	
URL address:	http://VHNAME.domain.com
Alternative URL addresses:	serveralias
Site administrator email:	administrator
FTP User:	ftpuser
Site folder location:	/home1/_sites/VHNAME/
Internet Availability:	No
Safe mode:	Disabled
Access authorized only to the directories:	
Character encoding:	
Always keep the typed URL:	Yes

Figura 4.31: *Virtual host* criado pela aplicação-teste

```
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere
```

Figura 4.32: Regra na *firewall* criada pela aplicação-teste

Forward zone	
master	domain.com
master	oidomain190.com

Figura 4.33: Zona DNS criada pela aplicação-teste

Show 10 entries Search:

Machines	
Name	IP
ipbrick190	172.31.3.190
ipbrick191	172.31.3.191

1 - 2 | Total: 2

Figura 4.34: Registos DNS do tipo A criados pela aplicação-teste

da interface gráfica foi necessário recorrer à linha de comandos para utilização do gestor de pacotes do sistema (neste caso, o utilitário dpkg) de maneira a efetuar a desinstalação do pacote instalado. Procedendo à remoção do pacote, o utilitário dpkg verifica a existência de ficheiros prerm e postrm integrados aquando a sua instalação e executa as ações aí definidas em caso afirmativo. Na figura 4.36 é possível identificar o início da desinstalação do pacote e na figura 4.37 verifica-se o *output* da chamada DELETE /bugfix, sinal de progresso do processo.

Após finalização foi possível verificar, como feito aquando a instalação da aplicação, a execução bem sucedida das eliminação de recursos especificadas revertendo o sistema para o estado pré-instalação desejado.

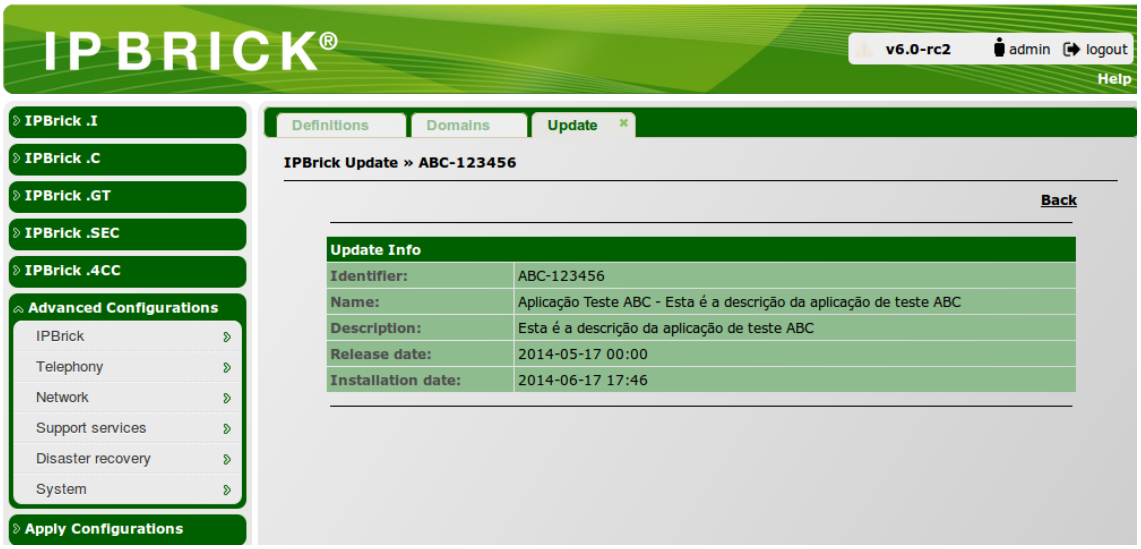

Figura 4.35: Registo *bugfix* da aplicação-teste

```
jsobreira189:~# dpkg -r "apiclient"
(Reading database ... 174614 files and directories currently installed.)
Removing apiclient ...
POST /INST <br />
HTTP Code: 200<br />
Response: <pre>array(2) {
  ["code"]=>
  array(1) {
    [0]=>
    int(200)
  }
  ["message"]=>
```

Figura 4.36: Inicio do processo de desinstalação da aplicação-teste

```
DELETE /BUGFIX <br />
HTTP Code: 200<br />
Response: <pre>array(2) {
  ["code"]=>
  array(1) {
    [0]=>
    int(200)
  }
  ["message"]=>
  array(1) {
    [0]=>
    string(59) "Application ABC-123456 entry removed from the bugfixes list"
  }
}
</pre><br /><br />
```

Figura 4.37: Excerto do *output* do processo de desinstalação da aplicação-teste

4.5.2 Aplicações IPBrick

Para o teste final de integração de uma aplicação no sistema IPBrick foi disponibilizada uma aplicação utilizada pela empresa em ambientes de *Call Center* denominada *callcenter4ipbrick*. Após análise do pacote

foi possível identificar a necessidade de criação de um *virtual host* e de um registo CNAME, juntamente com o habitual registo na lista de *bugfixes*. A adaptação do código do pacote para a utilização da API de integração acabou por ser tão simples como a substituição dos excertos de acesso à base de dados do sistema pelas chamadas apresentadas na aplicação-teste (4.5.1 e adição de chamadas de inicio e final de instalação, sendo que estas substituiriam algumas verificações e processos efetuados pelo código como é o caso da comparação versão do *software* IPBrick necessária com a instalada e criação de base de dados de configurações temporárias. No entanto, ao tentar instalar a aplicação no sistema fomos alertados pela própria API de uma situação que impossibilitaria a sua instalação: uma dependência não resolvida (figura 4.38).

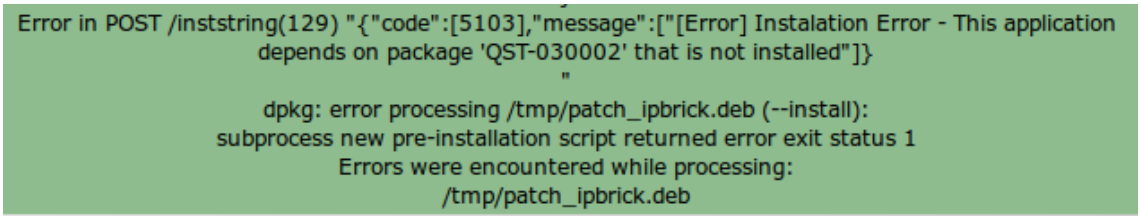

Figura 4.38: Alerta de erro de dependência durante a instalação da aplicação *callcenter4ipbrick*

Com o intuito de garantir uma aplicação funcional, foi requisitada à IPBrick o pacote em falta que rapidamente foi adaptado e instalado no sistema, sendo que este não necessitaria de utilizar funcionalidades da API excetuando os essenciais registo de *bugfixes* e pedidos relativos ao processo de instalação. A grande necessidade deste pacote prendia-se com a criação de uma base de dados utilizada pela aplicação *callcenter4ipbrick*. Com a base de dados criada (figura 4.39) procedeu-se a uma nova tentativa de instalação da aplicação de *call center*.

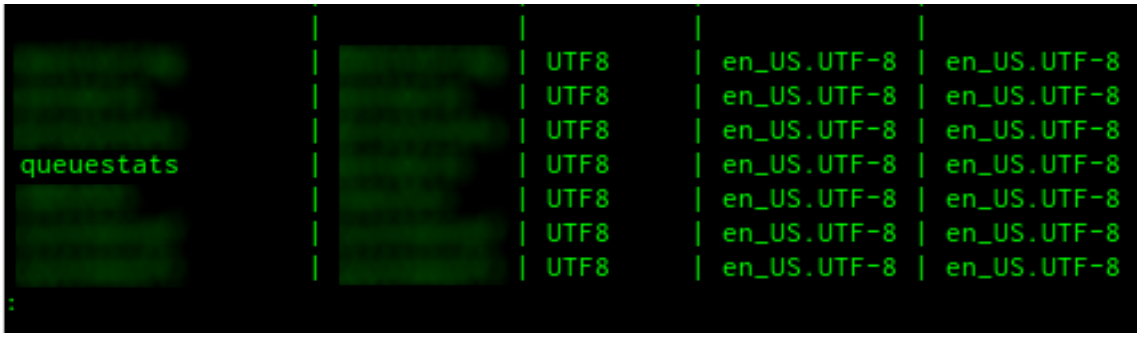

Figura 4.39: Base de dados *queuestats*

Não sendo apresentados erros (figuras 4.40 e 4.41) foi então possível confirmar o sucesso da instalação da aplicação no sistema.
Ao verificar a correta criação de um *virtual host* para a página da aplicação (figura 4.42), procedeu-se à integração de um posto no domínio do servidor IPBrick e, através de um *browser*, foi tentado o acesso à página *callcenter.domain.com*.
Como é possível visualizar na figura 4.43 é identificada a existência de uma página à qual se pretende aceder, no entanto possíveis problemas de compatibilidade de versões impossibilitaram a disponibilização correta da página isto porque não foi executada qualquer tipo de modificação de secções do código ainda otimizadas para a versão 5.2 do sistema IPBrick mas sem relevância para a integração da aplicação.

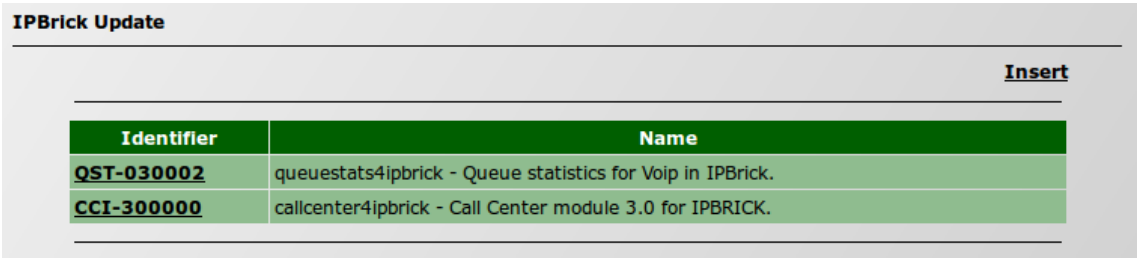

Figura 4.40: Lista de *bugfixes* após instalação da aplicação *callcenter4ipbrick*

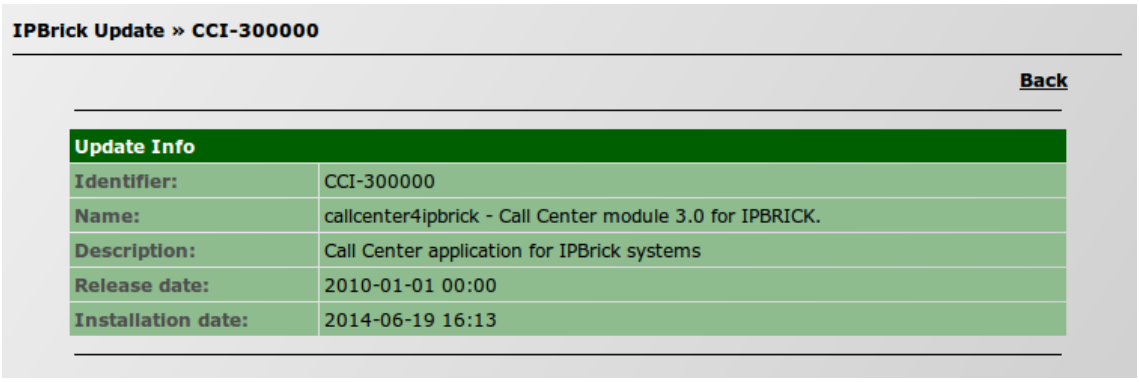

Figura 4.41: Detalhes da entrada de *bugfixes* registada pela aplicação *callcenter4ipbrick*

Figura 4.42: Lista de *virtual hosts* do sistema após instalação da aplicação *callcenter4ipbrick*

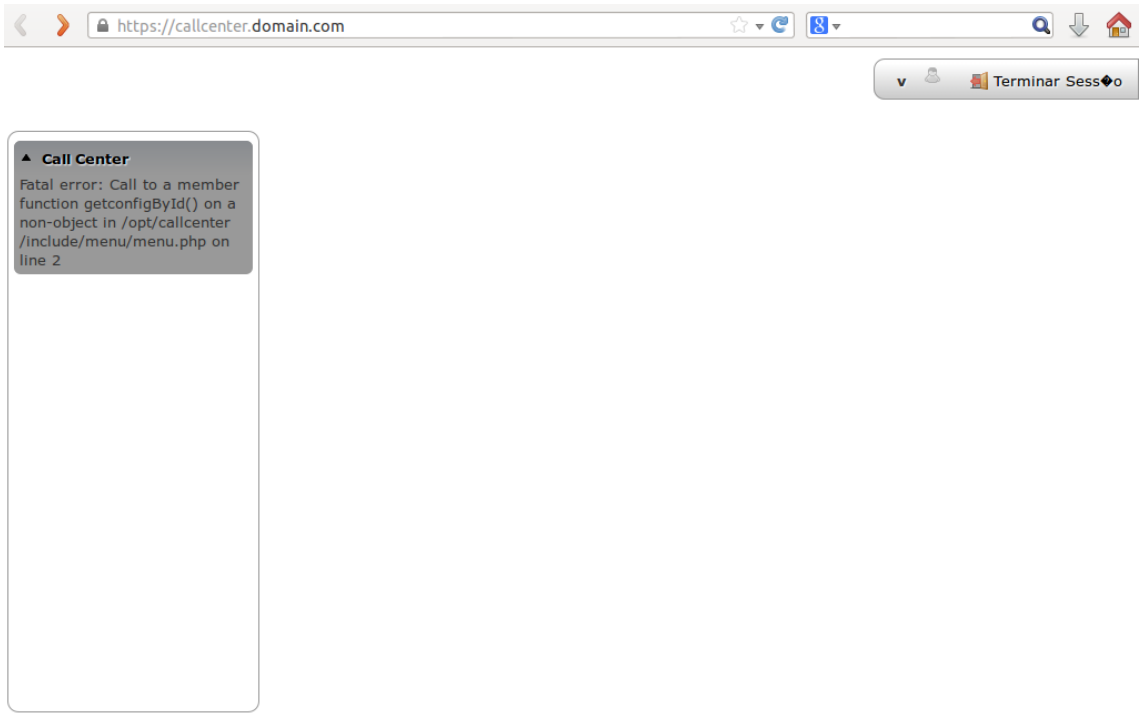

Figura 4.43: Acesso à página *callcenter.domain.com*

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Satisfação dos Objetivos

No final deste projeto foi possível formalizar uma nova estrutura robusta, escalável e atual e apresentar uma solução básica e funcional para o problema apresentado. Foram cobertas as funcionalidades mais importantes e elaborado um modelo possível de ser utilizado para adaptação de aplicações a esta solução. No que toca a objetivos propostos é sensato afirmar que foram cumpridos. O produto final, como apresentado neste documento, foi capaz de executar aquilo que era expectável mesmo sendo utilizadas aplicações com um grau de complexidade já elevado. No entanto as caraterísticas desta solução garantem mais do que o proposto como é o caso de oferecer a possibilidade de desenvolver esquemas de desinstalação além dos de instalação. Os seus métodos de segurança juntamente com uma arquitetura simples, versátil e recente tornam viável a sua adaptação para um ambiente de redes externas. Todos os detalhes da solução foram desenvolvidos com o intuito de garantir liberdade aos desenvolvedores externos mantendo o controlo necessário internamente, tendo sido essa a preocupação principal demonstrada no inicio do projeto. É, sobretudo, além de uma solução para o problema atual apresentado e abordado neste projeto, um ponto de partida que não só não limita o desenvolvimento de um produto de maior escala como incentiva a essa progressão.

5.2 Trabalho Futuro

A fase de integração de uma aplicação com a API desenvolvida apresentou muitas ideias de possíveis melhorias à solução aqui apresentada. A sua estrutura já garante o controlo de acesso por via de diferentes mecanismos, mas é possível e desejável a divisão de funcionalidades da API em grupos distintos além da simples rejeição ou aceitação do acesso à mesma. Existe também alguma falta de controlo pela API sobre as ações dos seus utilizadores. Embora contra a ideologia da arquitetura REST, seria útil a associação de um pedido não só a um processo de instalação mas também a uma aplicação. Isto permitiria o controlo sobre os recursos manipulados, podendo ser em certos casos impostas restrições ao acesso apenas a recursos criados pela própria aplicação. A gestão das aplicações já instaladas também não é a ideal, sendo importante a identificação de pacotes instalados pelo gestor de pacotes do sistema e de nomes de pacotes sem informação exata. Todas estas funcionalidades não necessitariam de modificações em larga escala para serem implementadas, não tendo sido incluídas no produto final aqui apresentado apenas por restrições de tempo ou identificação tardia. No que toca a alterações mais demoradas, a reavaliação de certos mecanismos já utilizados pelo sistema IPBrick e aproveitados para elaboração desta solução seria a maior prioridade, como

é o caso da gestão da base de dados de configurações temporárias que se apresentou como um *bottleneck* em termos de eficácia do processo de instalação de uma aplicação. Acima de tudo, é possível identificar uma grande margem de expansão para este produto. As suas funcionalidades apenas se encontram limitadas pelas próprias funcionalidades do sistema e a adaptação e união de uma API de comunicação REST à solução apresentada poderia ser uma opção válida, principalmente com um controlo de grupos de acesso, apresentando uma vasta gama de novas possíveis funcionalidades como é o caso da comunicação e controlo com o sistema através de dispositivos móveis.

Anexo A

Tabela detalha de requisitos da solução

		Biblioteca(s) utilizada(s)	**Tabela(s) utilizada(s)**	**Outros recursos utilizados**
A	**Processo de Instalação**			
1	**Estado da Instalação**			
1.1	Identificação do estado da instalação ("a decorrer"/ "não inicializada")			Identificação de existência de ficheiro temporário
2	**Início da Instalação**			
2.1	Validação de dados			
2.1.1	Verificação de compatibilidade da versão IPBrick		ipbrickconfig	
2.1.2	Verificação de dupla instalação		bugfixes	
2.1.3	Verificação de dependências		bugfixes	
2.1.4	Verificação de conflitos		bugfixes	
2.2	Activação de flag de instalação			Criação de ficheiro temporário
3	**Finalização / Cancelamento da Instalação**			
3.1	Desactivação da flag de instalação			Eliminação de ficheiro temporário
B	**Gestão de Utilizadores**			
1	**Informação sobre utilizador**	LibUsers		
2	**Criação de utilizador**	LibUsers		
2.1	Criação de conta de e-mail			
3	**Alteração de dados de utilizador**	LibUsers		
4	**Eliminação de utilizador**	LibUsers		
C	**Gestão de Firewall**			
1	**Informação sobre regra(s) na firewall**	*LibFirewall*	firewall	
	Filtragem por parâmetros (protocolo / porta / ip de destino / ip de origem)	*LibFirewall*	firewall	

2	**Adição de regra na firewall**	*LibFirewall*	firewall	
3	**Eliminação de um regra na firewall**	*LibFirewall*	firewall	
D	**Gestão de Virtual Hosts**			
1	**Informação sobre Virtual Host**	*LibVH*	apache	
			named_conf	
			dns_cname	
1.1	identificação dos parâmetros de configuração associados ao Virtual Host	*LibVH*	apacheotheropt	
1.2	Identificação das contas FTP associadas a um Virtual Host	*LibVH*	ftp_accounts	
2	**Criação de Virtual Host**	*LibVH*	apache	
			servidor	
			named_conf	
2.1	Definição de parâmetros de configuração do Virtual Host	*LibVH*	apacheotheropt	
2.2	Adição de registo CNAME	*LibVH*	dns_cname	
2.3	Criação de conta FTP	*LibVH*	ftp_accounts	
3	**Modificação de Virtual Host**	*LibVH*	apache	
			named_conf	
3.1	Modificação dos parâmetros de configuração do Virtual Host	*LibVH*	apacheotheropt	
3.2	Modificação do registo CNAME	*LibVH*	dns_cname	
3.3	Adição de nova conta de acesso FTP	*LibVH*	ftp_accounts	
4	**Eliminação de Virtual Host**	*LibVH*	apache	
			named_conf	
			apacheotheropt	
4.2	Eliminação do registo CNAME	*LibVH*	dns_cname	
4.3	Eliminação das contas de acesso FTP associadas	*LibVH*	ftp_accounts	
E	**Gestão de registos DNS**			
1	**Informação sobre registo DNS**	***LibDNS***		
1.1	Informação sobre registos A	*LibDNS*	dns_in_a	
1.2	Informação sobre registos PTR	*LibDNS*	dns_in_ptr	
1.3	Informação sobre registos MX	*LibDNS*	dns_in_mx	
1.4	Informação sobre registos NS	*LibDNS*	dns_in_ns	
1.5	Informação sobre registos TXT	*LibDNS*	dns_in_txt	
1.6	Informação sobre registos SRV	*LibDNS*	dns_in_srv	
2	**Criação de registo DNS**			
2.1	Criação de registo A	*LibDNS*	dns_in_a	
2.2	Criação de registo PTR	*LibDNS*	dns_in_ptr	
2.3	Criação de registo MX	*LibDNS*	dns_in_mx	
2.4	Criação de registo NS	*LibDNS*	dns_in_ns	
2.5	Criação de registo TXT	*LibDNS*	dns_in_txt	

2.6	Criação de registo SRV	*LibDNS*	dns_in_srv	
3	**Modificação de registo DNS**			
3.1	Modificação de registo A	*LibDNS*	dns_in_a	
3.2	Modificação de registo PTR	*LibDNS*	dns_in_ptr	
3.3	Modificação de registo MX	*LibDNS*	dns_in_mx	
3.4	Modificação de registo NS	*LibDNS*	dns_in_ns	
3.5	Modificação de registo TXT	*LibDNS*	dns_in_txt	
3.6	Modificação de registo SRV	*LibDNS*	dns_in_srv	
4	**Eliminação de registo DNS**			
4.1	Eliminação de registo A	*LibDNS*	dns_in_a	
4.2	Eliminação de registo PTR	*LibDNS*	dns_in_ptr	
4.3	Eliminação de registo MX	*LibDNS*	dns_in_mx	
4.4	Eliminação de registo NS	*LibDNS*	dns_in_ns	
4.5	Eliminação de registo TXT	*LibDNS*	dns_in_txt	
4.6	Eliminação de registo SRV	*LibDNS*	dns_in_srv	
F	**Gestão de entradas na lista de bugfixes**			
1	**Informação sobre entrada na lista de bugfixes**	*LibBugfixes*	bugfixes	
2	**Adição de entrada na lista de bugfixes**	*LibBugfixes*	bugfixes	
3	**Modificação de entrada na lista de bugfixes**	*LibBugfixes*	bugfixes	
4	**Eliminação de entrada na lista de bugfixes**	*LibBugfixes*	bugfixes	
G	**Geral**			
1	**Identificação da origem do pedido**			Controlador de Interfaces
2	**Autenticação dos pedidos**			Controlador de Interfaces
3	**Verificação da ligação à base de dados**			Controladores
4	**Apresentação de erros com mensagens informativas quanto à sua origem**			Controlador de Interfaces + erros IPBrick
5	**Registos de acesso em ficheiro de texto**			Escrita em ficheiro
6	**Registos de erros em ficheiro de texto**			Escrita em ficheiro

Tabela A.1: Tabela de requisitos da solução

Anexo B

Exemplo de um ficheiro WSDL

É apresentado neste anexo a estrutura de um documento WSDL segunda a *World Wide Web Consortium*.

```
1
2  <wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
3
4      <import namespace="uri" location="uri"/>*
5
6      <wsdl:documentation .... /> ?
7
8      <wsdl:types> ?
9          <wsdl:documentation .... />?
10         <xsd:schema .... />*
11         <-- extensibility element --> *
12     </wsdl:types>
13
14     <wsdl:message name="nmtoken"> *
15         <wsdl:documentation .... />?
16         <part name="nmtoken" element="qname"? type="qname"?/> *
17     </wsdl:message>
18
19     <wsdl:portType name="nmtoken">*
20         <wsdl:documentation .... />?
21         <wsdl:operation name="nmtoken">*
22             <wsdl:documentation .... /> ?
23             <wsdl:input name="nmtoken"? message="qname">?
24                 <wsdl:documentation .... /> ?
25             </wsdl:input>
26             <wsdl:output name="nmtoken"? message="qname">?
27                 <wsdl:documentation .... /> ?
28             </wsdl:output>
29             <wsdl:fault name="nmtoken" message="qname"> *
30                 <wsdl:documentation .... /> ?
31             </wsdl:fault>
```

```
32          </wsdl:operation>
33      </wsdl:portType>
34
35      <wsdl:binding name="nmtoken" type="qname">*
36          <wsdl:documentation .... />?
37          <-- extensibility element --> *
38          <wsdl:operation name="nmtoken">*
39              <wsdl:documentation .... /> ?
40              <-- extensibility element --> *
41              <wsdl:input> ?
42                  <wsdl:documentation .... /> ?
43                  <-- extensibility element -->
44              </wsdl:input>
45              <wsdl:output> ?
46                  <wsdl:documentation .... /> ?
47                  <-- extensibility element --> *
48              </wsdl:output>
49              <wsdl:fault name="nmtoken"> *
50                  <wsdl:documentation .... /> ?
51                  <-- extensibility element --> *
52              </wsdl:fault>
53          </wsdl:operation>
54      </wsdl:binding>
55
56      <wsdl:service name="nmtoken"> *
57          <wsdl:documentation .... />?
58          <wsdl:port name="nmtoken" binding="qname"> *
59              <wsdl:documentation .... /> ?
60              <-- extensibility element -->
61          </wsdl:port>
62          <-- extensibility element -->
63      </wsdl:service>
64
65      <-- extensibility element --> *
66
67  </wsdl:definitions>
```

Referências

[1] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pœ l Krogdahl, Min Luo, e Tony Newling. *Patterns: Service-Oriented Architecture and Web Services*, volume 1 de *IBM Redbooks*. IBM Corp., International Technical Support Organization, 2004. URL: http://www.chinagrid.net/grid/paperppt/Patterns-Services.pdf, doi:10.1109/SOSE.2005.5.

[2] Xinyang Feng e Ying Fan. REST: An alternative to RPC for Web services architecture. *2009 First International Conference on Future Information Networks*, páginas 7–10, Outubro 2009. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5339611, doi:10.1109/ICFIN.2009.5339611.

[3] Gavin Mulligan e D Gracanin. A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. *Simulation Conference (WSC), ...*, 2009. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5429290.

[4] G. Krasner e S. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object Oriented Programming*, páginas 26–49, 1988.

[5] Programmable Web. API Directory - ProgrammableWeb. [Accessed 16 February 2014]. URL: http://www.programmableweb.com/apis/directory.

[6] Nick Gall. WOA: Putting the Web Back in Web Services, 2008. [Accessed 30 January 2014]. URL: http://blogs.gartner.com/nick_gall/2008/11/19/woa-putting-the-web-back-in-web-services/.

[7] Alex Ng, Shiping Chen, e Paul Greenfield. An evaluation of contemporary commercial SOAP implementations. Em *5th Australasian Workshop on Software and System Architectures (AWSA 2004)*, páginas 64–71, Melbourne, Vic., 2004. Swinburne Uni. Tech. URL: http://mercury.it.swin.edu.au/ctg/AWSA04/awsa04-proc.pdf#page=72.

[8] D. Crockford. Request For Comments 4627 - The application/json Media Type for JavaScript Object Notation (JSON). URL: http://www.ietf.org/rfc/rfc4627.txt.

[9] Hao He. What Is Service-Oriented Architecture. *XML.com*, páginas 1–5, 2003.

[10] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, e Clemente Izurieta. Comparison of JSON and XML Data Interchange Formats: A Case Study. *CAINE*, 2009:157–162, 2009.

[11] World Wide Web Consortium. Web Services Architecture. [Accessed 16 February 2014]. URL: http://www.w3.org/TR/ws-arch/.

[12] Pavan Kumar Potti. *On the Design of Web Services: SOAP vs. REST*. Tese de doutoramento, University of North Florida, 2011. URL: http://digitalcommons.unf.edu/etd/138/.

[13] Nan-Chao Huang. *A Cross Platform Web Service Implementation Using SOAP*. Tese de doutoramento, Knowledge Systems Institute, 2003. URL: http://www.ksi.edu/thesis/rhuang/rhuang.pdf.

[14] Microsoft Corporation. Understanding SOAP. [Accessed 2 February 2014]. URL: http://msdn.microsoft.com/en-us/library/ms995800.aspx.

[15] Microsoft Corporation. Understanding WSDL. [Accessed 2 February 2014]. URL: http://msdn.microsoft.com/en-us/library/ms996486.aspx.

[16] World Wide Web Consortium. Web Service Definition Language (WSDL). [Accessed 6 February 2014]. URL: http://www.w3.org/TR/wsdl.

[17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Phd, University of California, 2000.

[18] J. Franks, P Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, e L. Stewart. Request For Comments 2617 - HTTP Authentication: Basic and Digest Access Authentication. URL: https://www.ietf.org/rfc/rfc2617.txt.

[19] D. Hardt. Request For Comments 6749 - The OAuth 2.0 Authorization Framework. URL: http://tools.ietf.org/html/rfc6749.

[20] H. Krawczyk, M. Bellare, e R. Canetti. Request For Comment 2104 - HMAC: Keyed-Hashing for Message Authentication. URL: http://www.ietf.org/rfc/rfc2104.txt.