IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE ESTÁGIO

Licenciatura em Engenharia Informática

Eduardo Manuel Fernandes Dias

dezembro | 2015

ARQUITECTURA DE APIS

Projecto de Informática realizado em contexto de estágio

Engenharia Informática 2014/2015

Eduardo Manuel Fernandes Dias

Nº 1010607

Supervisor Dom Digital: António Matias Gil

Orientador: José Carlos Fonseca

Dezembro de 2015

Ficha de identificação

Aluno	
Nome	Eduardo Manuel Fernandes Dias
Número	1010607
Contacto	eduardogrd@gmail.com

Instituição de acolhimento do estágio	
Nome	Dom Digital – Novas Tecnologias de Informação, Lda.
Morada	Av. Rainha D. Amélia, 142 Cave
Código-postal	6300-749
Localidade	Guarda
Telefone	271224509
Fax	271230699
Data de início	2015/06/15
Data de fim	2015/09/11

Supervisor	
Nome	António Matias Gil
Grau académico	--

Docente orientador	
Nome	José Carlos Coelho Martins Fonseca
Grau académico	Doutor
Contacto	josefonseca@ipg.pt

Resumo

Este documento descreve o projecto realizado em contexto de estágio no âmbito da unidade curricular Projecto de Informática, na Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O conceito *Application Programming Interface* (API) tem sido muito utilizado no desenvolvimento de aplicações como estratégias comerciais para fornecimento de serviços *online.* Para além de serem suportadas por praticamente todo o tipo de dispositivos com capacidade de acesso à Internet, a versatilidade das APIs permite modelar qualquer género de negócio. O estado actual da tecnologia *cloud* permite o desenvolvimento, lançamento e gestão cada vez mais simplificado deste género de serviços, possibilitando a abstração da montagem, instalação e manutenção da infraestrutura de suporte.

O estágio foi proposto pela empresa Dom Digital, com o propósito de implementar e gerir a Ardina.API, uma API para armazenamento e distribuição de conteúdos multimédia. É suportada pela infraestrutura Force.com da Salesforce e gerida com os serviços providenciados pela 3scale num servidor alojado na Heroku. Foi implementado um novo recurso, *tags*, que confere duas novas funcionalidades à API: devolve todas as palavras-chave associadas ao conteúdo ou uma palavra-chave identificada por uma chave fornecida em parâmetro que a identifica univocamente. Instalou-se o serviço *Cross-Origin Resource Sharing* para contornar a *Same Origin Policy*, permitindo que sejam efectuados pedidos à Ardina.API a partir de *browsers* numa origem diferente daquela do domínio onde se encontra alojada a API. Para reforçar a segurança da API, restringiu-se todo o tráfego recebido a HTTPS. Embora o servidor na Heroku já fornecesse serviços de *logging*, foi instalado um *add-on* que guardasse mais *logs* e fornecesse funcionalidades de notificação no caso de acontecerem eventos específicos, como a ocorrência de picos de tráfego. Por último, avaliou-se o serviço da Ardina.API com testes funcionais, onde foram encontrados e corrigidos erros, e a sua disponibilidade para responder a pedidos.

Neste relatório, é feito o enquadramento da Ardina no mercado de conteúdos multimédia e são descritas as tarefas realizadas durante o estágio.

Palavras-chave: API, REST, RESTful API, JSON, Salesforce, Heroku, 3scale

Abstract

This document describes the project carried out during an internship at Dom Digital, part of the IT Project curricular unit of the Computer Engineering Degree at the Higher School of Technology and Management (Escola Superior de Tecnologia e Gestão), of the Polytechnic Institute of Guarda (Instituto Politécnico da Guarda).

The Application Programming Interface (API) concept has been widely used to develop applications as commercial strategies for distributing online services. Besides being supported by every kind of device with Internet access capabilities, the APIs' versatility allows the modelling of a great diversity of businesses. Current cloud technology allows for simpler development, deployment and management of these services, abstracting assembly, installation and upkeep of the supporting infrastructure.

Dom Digital proposed the internship with the purpose to implement and manage Adina.API, an API to host and distribute multimedia content. It is supported by the Force.com infrastructure, with management services provided by 3scale from a server hosted in Heroku. In this report, Ardina is contextualized in the multimedia content market, and the several tasks performed during the internship are described. A new resource, tags, was implemented, giving two new functionalities to the API: it returns all keywords associated to the content or a single keyword uniquely identified by a key provided through parameters. The Cross-Origin Resource Sharing service was installed to bypass the Same Origin Policy, allowing for requests to be made to Ardina.API from browsers in a different origin from the one where Ardina.API is hosted from. To boost the API's security, all traffic has been redirected to HTTPS. Although the server on Heroku already provided logging services, an add-on has been installed to save more logs and provide notification functionalities, in the case of occurrance of specific events, like traffic-spikes. Lastly, Ardina.API's service was evaluated with functional tests, through which errors were found and solved, and its availability to answer requests.

In this report, Ardina is contextualized in the multimedia content market and the several tasks performed in the internship are described.

Keywords: API, REST, RESTful API, JSON, Salesforce, Heroku, 3scale

Do. Or do not. There is no try.

- Yoda

Índice

Índice de figuras

Índice de tabelas

1 Introdução

Este documento descreve o trabalho realizado pelo aluno Eduardo Manuel Fernandes Dias durante o seu estágio na Dom Digital – Novas Tecnologias de Informação Lda. e serve como objecto de avaliação para a unidade curricular Projecto de Informática decorrida no ano lectivo de 2014/2015, pertencente à Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

1.1 Motivação

Uma *Application Programming Interface* (API) é uma camada de abstração da implementação de um sistema, um *interface* que define uma série de regras para comunicar com outras aplicações de *software*, com o objectivo de simplificar a troca de dados e a disponibilização de serviços. Na *web*, a utilização das APIs tem crescido exponencialmente nos últimos anos (ProgrammableWeb). Tendo em conta o facto de serem suportadas em todo o género de dispositivos com capacidades de acesso à Internet, têm sido aplicadas como soluções estratégicas para distribuição de serviços e conteúdos em diversas áreas de negócio. De momento, são mais utilizadas em aplicações de mapas, serviços de viagens e distribuição e *streaming* de música (ProgrammableWeb). Com os avanços da tecnologia *cloud*, as *Platform as a Service* (PaaS), serviços de *cloud computing* que "permitem que os seus utilizadores criem aplicações de *software* utilizando as ferramentas fornecidas pelo fornecedor" (Interoute), providenciam uma infraestrutura robusta e central a partir da qual os programadores podem desenvolver, lançar e gerir as suas APIs, abstraindo-se da instalação, configuração e manutenção do suporte físico do *back-end* (Wikipedia). Dado que a Dom Digital é especializada na produção de serviços *web*, pretende aproveitar as potencialidades que as APIs proporcionam para desenvolver um serviço *web* onde canais de TV, jornais e difusoras de rádio possam armazenar e distribuir os seus conteúdos multimédia: a Ardina.API (Ardina). Para esta API podem ser criadas aplicações móveis e páginas *web* que consumam os seus recursos e os apresentem aos seus utilizadores. Neste sentido, a Dom Digital propôs a

implementação de novas funcionalidades e gestão desta API como tema de projecto a realizar em contexto de estágio para os alunos finalistas da Licenciatura em Engenharia Informática.

Este é um projecto aliciante, dada a sua interdisciplinaridade, na medida em que é necessário acompanhar as várias fases no processo do desenvolvimento de um produto de *software*, desde a implementação das novas funcionalidades utilizando tecnologias muito actuais e de elevada importância no sector, passando pelos testes e terminando com o lançamento *online* da API. Sendo um projecto em contexto de estágio, apresentou-se também como uma excelente primeira oportunidade para ganhar experiência numa situação de trabalho em ambiente empresarial.

1.2 Objectivos

O estágio tem como objectivos principais o domínio do conceito de API aplicado à indústria dos média e a melhoria da Ardina.API, compreendendo os conceitos e tecnologias associadas através da realização das seguintes tarefas:

- Contribuição para o desenvolvimento de uma aplicação Android, adaptada a partir da aplicação RTP Play já desenvolvida pela Dom Digital, para visualização de conteúdo áudio-visual *live* e *on-demand* de um cliente seu. Desta forma, é posta em prática a utilização dos recursos de uma API na óptica do seu consumo por parte do programador.
- Implementação de novas funcionalidades e gestão da Ardina, que é uma API para conteúdos multimédia.

Com a realização das actividades associadas às tarefas descritas, pretende-se ficar a conhecer:

- A arquitectura e o funcionamento do *back-end* de uma API, ou seja, o processamento de pedidos, a extracção de informação da base de dados e a construção de uma resposta devidamente formatada.
- Métricas de avaliação da API, como sejam a segurança, *performance*, disponibilidade e escalabilidade no lançamento e disponibilização do serviço numa PaaS.
- Actividades de gestão, ligadas ao controlo de acesso à API e monitorização do tráfego de rede gerado.

- Usabilidade, isto é, tendo em conta que os utilizadores da Ardina são programadores, o serviço deverá ser desenvolvido e documentado para que percebam facilmente como é que as suas aplicações e páginas *web* podem consumir os recursos da API.

1.3 Estrutura do documento

Para além deste capítulo introdutório, o relatório contém mais seis. No capítulo 2 é feito um estudo do estado da arte, descrevendo a estrutura de uma API RESTful, como é o caso da Ardina, assim como a metodologia que habitualmente é aplicada na sua implementação. No capítulo 3 é descrita a arquitectura da Ardina e as tecnologias que a compõem. No capítulo 4 encontra-se a metodologia utilizada durante o estágio e apresenta-se o mapa de Gantt com as tarefas realizadas. No capítulo 5 são descritas as tarefas executadas para expandir as funcionalidades da Ardina. No capítulo 6 são discutidos os testes realizados para avaliar o funcionamento correcto da Ardina. Por fim conclui-se o relatório com uma análise do que foi realizado e considerações sobre o que poderá ser o futuro do desenvolvimento da Ardina.

2 Estado da arte

O termo *Application Programming Interface* (API) consiste num "conjunto de regras e especificações que um programa de software deve seguir para aceder e utilizar os serviços e recursos proporcionados por outro programa de software que implementa essa mesma API. Funciona como um interface entre diferentes programas e facilita as suas interacções, de uma forma semelhante à que o user interface facilita a interacção entre humanos e computadores" (3scale Networks S.L., 2011). Ou seja, uma API permite que uma aplicação seja desenvolvida de uma forma modular, acedendo a dados e/ou funcionalidades de outras aplicações. Existem APIs de vários tipos, como indicado na Tabela 1. A título de exemplo, a API da linguagem JavaScript permite a inclusão de bibliotecas (Javascripting) para expandir as suas funcionalidades e facilitar o desenvolvimento de programas, enquanto que a API da linguagem Java atinge o mesmo objectivo através de classes que, por sua vez, acrescentam funcionalidades proporcionadas pelos seus métodos (Oracle). Tratam-se de duas categorias diferentes de APIs, embora a ligação com as bibliotecas e as classes seja feita da mesma forma, que é através do código-fonte da aplicação.

Categoria	API
Serviços *web*	REST, SOAP, XML-RPC, JSON-RPC
Baseadas em bibliotecas	JavaScript, TWAIN
Baseadas em classes (orientadas a objectos)	Java API, Android API
Funções de Sistemas Operativos e rotinas	Acesso ao sistema de ficheiros e interface de utilizador
Hardware	Aceleração de vídeo da placa gráfica, acesso a discos rígidos

Tabela 1 - Categorização de APIs, adaptado de (Maddox, 2014).

Como serviço *web*, uma API disponibiliza os seus serviços através da Internet, da mesma forma que se acede a um *website*, ou seja, através de um enedereço que aponta para o domínio do serviço e onde se encontram parametrizados os recursos pretendidos. Age como intermediária na submissão dos pedidos ao seu *back-end* para que sejam processados, sendo,

também, responsável pela devolução de uma resposta estruturada com os resultados. Também é comum fornecer informação extra relativa ao estado do pedido, tais como, se o processamento foi realizado com sucesso, se são devolvidos dados, entre outras. Embora não exista nenhum padrão que especifique como é que uma API deve ser desenvolvida, o protocolo *Simple Access Object Protocol* (SOAP) e o estilo de arquitectura *REpresentational State Transfer* (REST) são os mais aplicados no seu desenvolvimento, dando origem a SOAP APIs e RESTful APIs, respectivamente. As SOAP APIs são mais utilizadas em ambiente empresarial ou em serviços que exijam um reforço na segurança, pois, tratando-se de um protocolo de comunicação, integra funcionalidades de autorização e segurança, embora se tratem de sistemas mais difíceis de implementar. As RESTful APIs são mais fáceis de implementar e seguem o mesmo estilo de arquitectura que a *World Wide Web*, sendo, por isso, mais utilizadas em aplicações distribuídas por este meio, como serviços dos media, redes sociais e *chat*. Este é um factor contribuinte para o sucesso das RESTful em comparação com as SOAP. Há ainda diferenças em termos de *interface*, na medida em que uma SOAP API "disponibiliza dados como um serviço (na forma de verbo e substantivo), por exemplo, *getUser* ou *PayInvoice*" (NordicAPIs), enquanto que uma RESTful API "disponibiliza dados como recurso (na forma de substantivo), por exemplo, *user* ou *invoice*" (NordicAPIs) (ProgrammableWeb) (Arcitura).

2.1 RESTful API

O estilo de arquitectura REST especifica um "conjunto de restrições para o *design* de componentes num sistema de *hypermedia* distribuído que poderá resultar numa arquitectura de *performance* mais elevado e de melhor manutenção" (Wikipedia).

A arquitectura deverá ser do tipo cliente-servidor, já que, tratando-se um *interface*, é necessário que o funcionamento interno dos seus componentes seja abstraído do *software* que requisita os seus recursos, promovendo o desenvolvimento independente de cada um. O serviço deverá ser *stateless*, ou seja, cada par pedido-resposta deverá ser independente, resultando em menos dados armazenados no servidor, cabendo ao cliente guardar informações relativas ao estado da sua sessão. No entanto, esta informação pode ser utilizada pelo servidor para controlo de acesso e persistência de dados em transições. Para evitar processamento extra, deverá permitir armazenar pedidos repetidos em *cache*, tanto pelo cliente como pelo servidor. O sistema deverá ser construído em camadas, permitindo a instalação de componentes adicionais e facilitando alterações à arquitectura, conferindo,

também, escalabilidade e redundância. O *interface* é "tipicamente, ausente de qualquer contexto de negócio porque, de forma a ser reutilizável por uma larga variedade de serviços e seus consumidores, necessita de providenciar capacidades genéricas e de alto-nível, abstractas o suficiente para acomodar um elevado número de requisitos para interacção entre serviços". Este é constituído por três elementos: métodos de acesso, tipo de media a aceder e a sintaxe de acesso ao recurso. Opcionalmente, o servidor poderá ter a capacidade de transferir código para ser executado na máquina do cliente, denominado de *code on demand* (Arcitura).

Uma RESTful API funciona exclusivamente sobre os protocolos HTTP e HTTPS. Possui um endereço base no qual está disponível o serviço e expõe os seus recursos através de *endpoints*, que nada mais são que pontos de ligação à API onde se encontram os seus recursos, especificados na URL. Recursos contêm colecções e elementos e são identificados através de um *Uniform Resource Identifier* (URI). Seguindo o exemplo da Tabela 2:

- **http://restfulapi.com** é o endereço base.
- **/itens** e **itens/1234** são ambos *endpoints,* pois especificam recursos diferentes, embora o segundo seja um elemento da colecção em "/itens".
- **itens** e **1234** são dois URIs que identificam dois recursos.

Deverá permitir operações de *Create, Read, Update* e *Delete* (CRUD) através dos verbos HTTP: PUT, GET, POST e DELETE. Cada verbo tem uma acção diferente consoante o recurso onde é utilizado, como mostra o exemplo da Tabela 2.

URL do pedido	Recurso acedido	POST (CREATE)	GET (READ)	PUT (UPDATE)	DELETE (DELETE)
http://restfulapi.com/itens	Colecção com os itens todos.	Cria novo item.	Mostra os itens todos.	Actualiza colecção inteira.	Apaga a colecção de itens.
http://restfulapi.com/itens/1234	Item 1234 da colecção itens.	Erro.	Mostra apenas este item.	Se existe, actualiza o item. Se não existe, dá erro.	Apaga este item.

Tabela 2 - Acções dos verbos HTTP numa RESTful API (adaptado de (Mulloy)).

Podem ainda ser utilizados parâmetros no URL para acções específicas sobre o recurso, o que não é recomendado, pois a complexidade de utilização aumenta com o número de parâmetros (Vaqqas).

2.2 Formatação da resposta

A resposta de uma RESTful API pode ser codificada nos vários formatos definidos no padrão *Multipurpose Internet Mail Extensions* (MIME) (Internet Assined Numbers Authority), no entanto, os mais utilizados são o *Javascript Object Notation* (JSON) e o *Extensible Markup Language* (XML) (ProgrammableWeb).

Embora tenha sido desenvolvido a partir da notação utilizada em Javascript para a descrição de objectos, o formato JSON é independente da linguagem e foi criado como alternativa ao XML, proporcionando uma sintaxe mais simples, com uma leitura mais fácil por humanos e máquinas, como mostra a Figura 1. Este facto tem vindo a contribuir para a crescente adopção de JSON em relação a XML, tendo já surgido casos de fornecedores que disponibilizam apenas este formato (Zazueta). No JSON, um objecto é uma colecção de pares chave-atributo, separados por ":" e definidos com os tipos *number*, *string*, *boolean* e *null*. Permite, também, a utilização de *arrays* para conjuntos ordenados de valores. Os caracteres "{}" e " []" delimitam objectos e *arrays*, respectivamente. Os diferentes pares chave-atributo são separados com o caracter ",". A notação simples e orientada a objectos facilita o mapeamento de JSON em objectos nas linguagens *Object Oriented Programming* (OOP), como seja o caso do Javascript (Bassett, 2015).

O XML é uma *markup* language, definida no padrão *XML 1.0 Specification* do W3C (World Wide Web Consortium). Tal como o HTML, utiliza *tags* para definir tipos de dados e elementos, ambos personalizáveis. Ao contrário do JSON, o XML possui um método de validação, o *XML Schema Definition* (XSD), permitindo que cliente e servidor garantam que os dados transmitidos se encontram devidamente formatados e definidos. Foi criada com o objectivo de fornecer um formato padrão para transmissão de dados entre aplicações. No entanto, e embora seja munido do XSD, processar um documento XML requer processamento extra relativamente ao JSON, já que o programa tem que processar, uma a uma, todas as linhas do documento recebido (Fawcett, 2012).

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Figura 1 - Comparação entre o mesmo conjunto de dados formatados em JSON e XML (W3Schools).

2.3 Implementação e lançamento

Uma API pode ser criada utilizando qualquer linguagem de programação. Tendo em conta o protocolo de comunicação utilizado, como foi mencionado na secção 2.1, deverá ser capaz de enviar e receber mensagens HTTP e HTTPS e suportar o formato de dados utilizado para codificar a resposta. No entanto, as linguagens mais utilizadas são PHP, Python e Ruby (DuVander, 2013). Muitos fornecedores criam bibliotecas *wrapper* e *software development kits* para diversas linguagens, com vista a facilitar a integração dos seus serviços através de métodos que simplificam a construção de pedidos e processamento de respostas, procurando maximizar o desenvolvimento de aplicações que consumam os seus recursos. Por exemplo, as APIs do YouTube, do Evernote e do GitHub são exemplos para os quais são fornecidas bibliotecas para PHP, Python, Ruby, Javascript, entre outras (Google Developers) (GitHub) (Evernote).

Uma PaaS é uma plataforma suportada por *cloud computing* que permite desenvolver, testar, efectuar tarefas de *debugging*, controlar versões, e lançar aplicações e/ou serviços *online*, ficando os fornecedores a cargo da gestão de toda a infraestrutura de suporte. Têm vindo a tornar-se na melhor forma de lançar uma API *online*, já que os custos e o trabalho envolvidos no seu lançamento são grandemente reduzidos, uma que vez que o cliente da PaaS não precisa de comprar *hardware*, nem de se preocupar com aspectos relacionados com a disponibilidade do serviço, como redundância e *uptime* (Kolb) (Interoute) (Linthicum). Muitas PaaS integram, também, ferramentas de gestão de APIs, como monitorização de tráfego, *developer portal*, entre outras. São exemplos a Apigee e a Amazon API Gateway (Apigee) (Amazon).

2.4 Gestão de APIs

Gestão de APIs trata-se do *processo de [as] publicar, promover e supervisionar num ambiente seguro e escalável. Também inclui a criação dos recursos de suporte que definem e documentam a API para o utilizador final* (Rouse). Embora seja possível implementar funcionalidades de gestão numa API, existem muitas empresas que fornecem plataformas dedicadas a estas tarefas. Uma boa plataforma de gestão deverá incluir (Apigee):

- Autenticação e segurança para registar e identificar utilizadores e assegurar confidencialidade de dados transmitidos. É comum a disponibilização de Oauth 1.0a e Oauth 2.0, dois *frameworks* de autenticação com base em *tokens*, e API Keys, uma *string* de caracteres aleatória ou personalizada que é passada nos argumentos de chamada da API (Stormpath, 2013).
- Monitorização e gestão de tráfego, através ferramentas de identificação e registo de erros ocorridos, a definição de limites do número de chamadas por unidade de tempo, entre outros.
- Armazenamento, para *cache* e outros dados necessários.
- *Developer portal*, que permita o registo de utilizadores, mostrar e documentar as funcionalidades da API, e oferecer-lhes suporte.
- Capacidades de monetização e análise de utilização, para os casos em que os serviços da API são vendidos, e ajudar a perceber como é que poderão ser melhorados para responder às necessidades dos utilizadores.

Os serviços de gestão são integrados no servidor da API de três formas diferentes (Psistakis, 2015):

- Instalando um *proxy* entre o servidor e o utilizador final, por onde passa todo o tráfego da API. Possibilita a utilização de *cache*, embora seja mais uma camada de complexidade adicionada à arquitectura.
- Um *agent*, implementado por *plugin* ao servidor da API. Embora esta solução seja mais leve que a anterior, exige que o *plugin* esteja implementado na mesma linguagem que é utilizada no servidor.
- *Proxy* e *agent*, permitindo a designação de tarefas específicas a cada um. Por exemplo, o *agent* pode ficar a cargo da autenticação enquanto que o *proxy* trata da *cache.*

3 Ardina.API

A Ardina é uma RESTful API que tem estado a ser desenvolvida pela Dom Digital. O seu propósito é proporcionar um serviço de armazenamento e distribuição de conteúdos multimédia, focado em recursos noticiosos apresentados em vários formatos (texto, áudio, imagens e vídeo) e direccionado para canais de televisão, jornais e difusoras de rádio que pretendam difundir os seus conteúdos através de *websites* e aplicações para *smartphones* e *tablets*. Encontra-se implementada totalmente na *cloud*, através das plataformas Salesforce e Heroku, e utiliza o serviço 3scale para tarefas de gestão.

Neste capítulo apresenta-se a arquitectura da Ardina e o papel de cada tecnologia integrada, descrevendo-se brevemente as empresas responsáveis por cada uma.

3.1 Salesforce

A Salesforce é proprietária da App Cloud, um produto PaaS suportado pela infraestrutura Force.com, que interpreta, executa e controla as linguagens proprietárias Apex e Visualforce, assim como o *Object-Oriented Database Management System* (OODBMS) integrado, também proprietário. Utilizando unicamente estas tecnologias é possível desenvolver, lançar e gerir serviços e páginas *web*. Possibilita integração com Heroku, para lançamento de aplicações que tratem os casos mais complexos. O foco da Salesforce é o seu serviço *Customer Relationship Management* (CRM), distribuído por diversas ferramentas de gestão de vendas, *marketing*, suporte a cliente, automação de casos e eventos, entre outros que estão indicados na Figura 2. A plataforma Salesforce é suportada por *cloud computing* e é totalmente acessível através do *browser*, não sendo necessário nenhum programa externo para utilizar as suas funcionalidades.

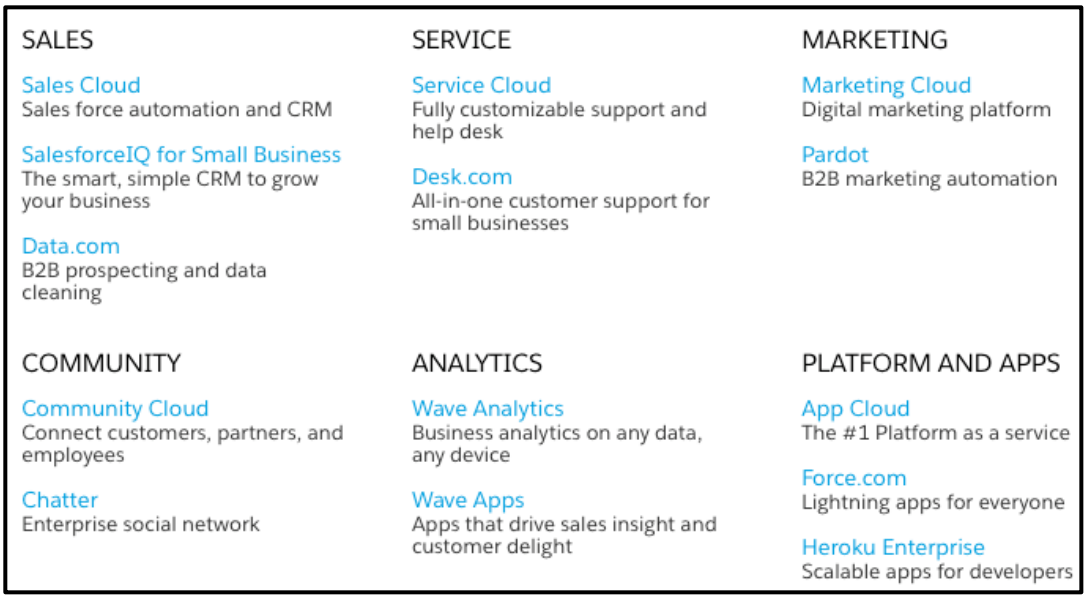

Figura 2 - Produtos CRM da Salesforce (Salesforce).

3.1.1 OODBMS

Os objectos na OODBMS da Salesforce são definidos utilizando tipos de dados já existentes noutras DBMS relacionais: campos de texto, números e numeração automática, datas e tempo, entre outros. Introduz alguns novos, como é o caso do tipo *formula*, que define campos que contêm o resultado de uma expressão fornecida, semelhante às funcionalidades que se encontram numa folha de cálculo. Todos os objectos têm que ter um nome, guardado no campo *name*, que os identifique, seja este texto ou número, embora não necessite de ser único. A chave que identifica cada objecto univocamente é o *ID*, que é designado automaticamente pela Force.com. Existem outros também definidos automaticamente, por exemplo, relacionados com a sua criação (*CreatedDate*, *CreatedById*) e acesso (*LastModifiedDate*).

Os objectos são relacionados de duas formas: *Master-Detail Relationship* e *Lookup Relationship*. A primeira utiliza-se em casos onde exista uma relação do tipo *parent-child* entre dois objectos (relação de muitos-para-um), guardando o ID do registo *parent* no campo *relationship field* do *child*. Desta forma, pode-se ter o campo *rollup summary field* no *parent*, que contém valores agregados a partir dos registos do objecto *child*. Apagando o objecto *parent*, todos os *child* são também apagados. A segunda relaciona dois objectos para relações um-para-um ou um-para-muitos.

As *queries* são construídas com a *Salesforce Object Query Language* (SOQL) e a *Salesfoce Object Search Language* (SOSL). A primeira é "semelhante ao comando SELECT

na Structured Query Language (SQL), (…) permitindo que seja especificado um objecto fonte (…), uma lista de campos a extrair e condições para seleccionar linhas no mesmo objecto (Salesforce). A sintaxe de uma instrução SOQL encontra-se na Figura 3.

```
1  SELECT one or more fields
2  FROM an object
3  WHERE filter statements and, optionally, results are ordered
```

Figura 3 - Sintaxe de uma instrução SOQL (Salesforce).

Ao contrário do SOQL, que só permite extrair informação de um objecto, o SOSL permite realizar pesquisas de texto em vários objectos de uma vez só. Utiliza o comando *FIND* e tem sintaxe diferente do SOQL, como se pode ver na Figura 4.

```
01 FIND {SearchQuery}
02 [ IN SearchGroup [ convertCurrency(Amount)] ]
03 [ RETURNING FieldSpec [ toLabel(fields)] ]
04 [ WITH DivisionFilter ]
05 [ WITH DATA CATEGORY DataCategorySpec ]
06 [ WITH SNIPPET[(target_length=n)] ]
07 [ WITH NETWORK NetworkIdSpec ]
08 [ LIMIT n ]
09
10 [ UPDATE [TRACKING], [VIEWSTAT] ]
```

Figura 4 - Sintaxe de uma instrução SOSL (Salesforce).

O *schema builder*, que se encontra na Figura 5, é o *interface* para navegar e realizar operações na sua OODBMS.

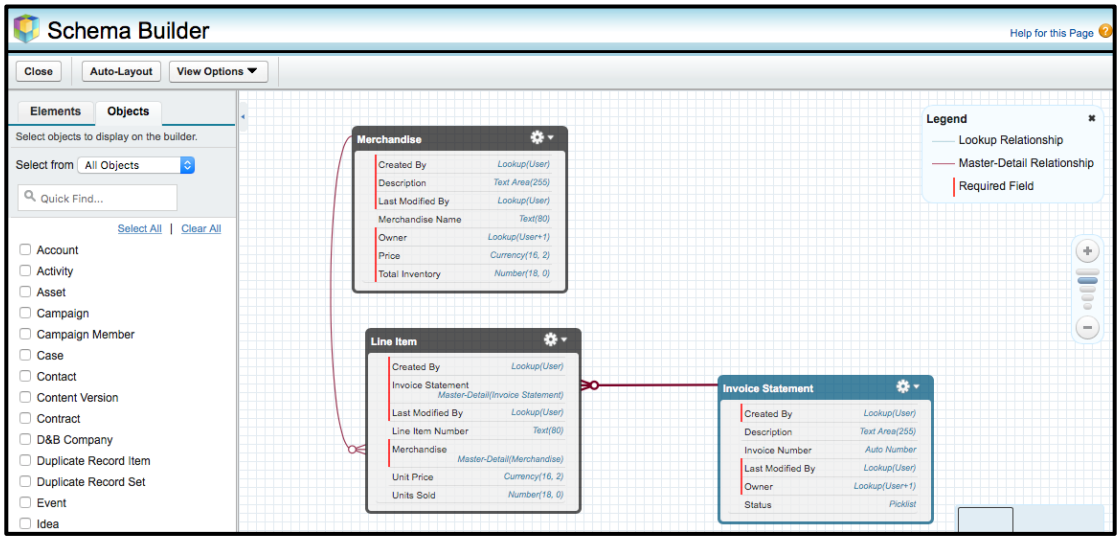

Figura 5 - O schema builder *é uma ferramenta para visualização de objectos na OODBMS da Salesforce.*

3.1.2 Apex

A Apex da Salesforce é uma linguagem de programação orientada a objectos (POO), com sintaxe semelhante a Java e desenvolvida a pensar na integração com as restantes tecnologias da plataforma Force.com. Desta forma, é possível utilizar instruções de *Data Manipulation Language* (DML), processar registos em massa com controlo transacional e criar *triggers* sem ser necessária a inclusão de bibliotecas e a configuração de ligações à base de dados. Para além disso, contém instruções normalmente encontradas neste género de linguagens, como controlo condicional (*if-else*), *looping* (*do-while*, *while* e *for*), invocação de métodos e tratamento de excepções. No código Apex da Figura 6 pode-se verificar como é fácil realizar operações sobre objectos da OODBMS:

- Na linha 1, instancia-se um novo objecto referenciando directamente o seu nome na OODBMS. O novo registo é criado com a instrução DML da linha 3.
- Na linha 6, os resultados da *query* são guardados na instância *inv*, através de uma instrução de atribuição, já que o *SELECT* é feito ao mesmo objecto. Nesta instrução pode verificar-se a vantagem de utilizar uma POO com uma OODBMS. A atribuição é directa, pois as classes existem dos dois lados.
- Na linha 10, altera-se um atributo e actualiza-se o registo da OODBMS na linha 11.
- Para apagar um registo, procede-se da mesma forma. Primeiro, selecciona-se o registo pretendido na linha 14 e executa-se o *delete* na sua instância.

```
01. Invoice_Statement__c inv = new Invoice_Statement__c(Description__c='My new invoice');
02. // Insert the invoice using DML.
03. insert inv;
04.
05. // First get the new invoice statement
06. Invoice_Statement__c inv = [SELECT Status__c
07.                             FROM Invoice_Statement__c
08.                             WHERE Description__c='My new invoice'];
09. // Update the status field
10. inv.Status__c = 'Negotiating';
11. update inv;
12.
13. // First get the new invoice statement
14. Invoice_Statement__c inv = [SELECT Status__c
15.                             FROM Invoice_Statement__c
16.                             WHERE Description__c='My new invoice'];
17. delete inv;
```

Figura 6 - Exemplo de código escrito em Apex (Salesforce).

As classes Apex criadas na Force.com ficam automaticamente preparadas para serem acedidas como serviços REST. Para isso, utiliza-se a notação *@RestResource* e define-se o

caminho através do qual pode ser acedida. A acção dos verbos HTTP é especificada com a notação *@Http<VERBO>* colocada antes da declaração do método. Por exemplo, um método *@HttpGet* é executado sempre que a classe recebe um pedido GET. A classe *RestContext* tem dois atributos: *request* contém os dados do pedido como a URI do recurso a aceder; *response* permite a construção da resposta, que também pode ser devolvida com a instrução *return*. Por omissão, os dados devolvidos são automaticamente convertidos em JSON. Se, por outro lado, for pretendida uma resposta XML, basta indicar o formato como parâmetro na *query*. Todos os serviços expostos desta forma têm a base URL do tipo *http://instancia-cliente.salesforce.com/services/apexrest* ao qual é incrementado o caminho especificado com a anotação *@RestResource* (Salesforce).

No exemplo da Figura 7, um pedido GET à URL *http://instancia-cliente.salesforce/services/apexrest/Merchandise/a04R00000007xX1IAI* devolve o resultado JSON ilustrado na Figura 8. A classe *MerchandiseManager* vai buscar o *ID* do recurso pretendido à *string* recebida no *RestContext.request*, como se pode ver na linha 6, e utiliza-o para extrair a informação necessária da base de dados na linha 8. A resposta é devolvida na linha 12 e automaticamente formatada em JSON.

```
01. @RestResource(urlMapping='/Merchandise/*')
02. global with sharing class MerchandiseManager {
03.     @HttpGet
04.     global static Merchandise__c getMerchandiseById() {
05.         RestRequest req = RestContext.request;
06.         String merchId = req.requestURI.substring(
07.                                     req.requestURI.lastIndexOf('/')+1);
08.         Merchandise__c result =
09.                         [SELECT Name,Description__c,Price__c,Total_Inventory__c
10.                          FROM Merchandise__c
11.                          WHERE Id = :merchId];
12.         return result;
13.     }
```

Figura 7 - Exemplo de uma classe Apex como serviço REST (Salesforce).

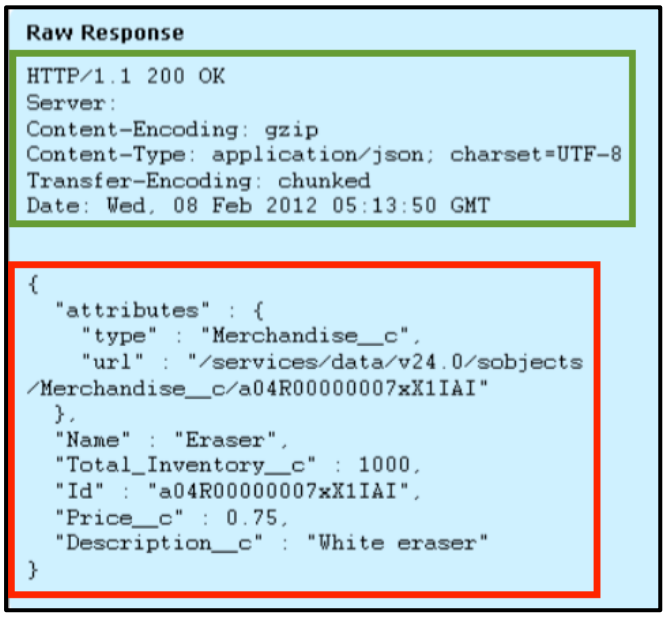

Figura 8 - Exemplo de uma resposta HTTP devolvida por uma classe Apex (Salesforce). O cabeçalho encontra-se a verde e o corpo da resposta, formatado em JSON, está a vermelho.

3.1.3 Classe ApiRest

A Ardina é exposta como uma RESTful API através da classe ApiRest. Actualmente está preparada para devolver imagens, áudios, vídeos, conteúdos (um "conteúdo" é qualquer elemento com contém dados mistos como texto, imagens, áudios e vídeos) e para realizar pesquisas sobre estes. Os dados podem ser filtrados por palavras-chave (*tag)* e data, fornecendo uma data inicial (*startDate*) e/ou uma final (*endDate*). Para casos em que o número de dados devolvidos é elevado, permite a paginação de resultados, bastando apenas indicar os parâmetros *pageSize* e *pageIndex*. A Figura 9 mostra todos os *endpoints* da Ardina.

Figura 9 - Endpoints *da Ardina.*

Na Figura 10 pode-se verificar a estrutura da classe *ApiRest*. No método *doGet* é processada a URI recebida assim como quaisquer parâmetros adicionais. Dependendo do pedido, será utilizado um ou mais métodos auxiliares para construir a resposta, que será uma instância da classe *ApiReturnClass*, construída, por sua vez, com base na *ApiDataClass*, que contém os dados requisitados à API, e a *ApiStatusClass*, onde são incluídos o código de estado e a respectiva mensagem associada. A informação é extraída da OODBMS utilizando os métodos auxiliares *get<something>*, por exemplo, *getImages*, *getFilesByTag*, entre outros.

```
@RestResource(urlMapping='/api/1/*')
global with sharing class ApiRest {

    @HttpGet
    global static ApiReturnClass doGet() {...
    }

    global class ApiReturnClass{...
    }

    global class ApiStatusClass{...
    }

    global class ApiDataClass{...
    }

    global static List<ApiFileClass> getImages(Map<String, String> params){...
    }

    global static List<ApiFileClass> getAudios(Map<String, String> params){...
    }

    global static List<ApiFileClass> getVideos(Map<String, String> params){...
    }

    global static List<ApiContentClass> getSearchContent(String query){...
    }

    global static ApiContentClass getSingleContent(Id reqId){...
    }

    global static List<ApiContentClass> getContents(Map<String, String> params){...
    }

    global static List<ApiContentClass> getListContents(Map<String, String> params){...
    }

    global static List<ApiContentClass> getContentsByTag(Map<String, String> params){...
    }

    public static Table getTableData(String recordId){...
    }

    global static ApiFileClass getSingleFile(String type, Id reqId){...
    }

    global static List<ApiFileClass> getFiles(String type, Map<String, String> params){...
    }

    global static List<ApiFileClass> getFilesByTag(String type, Map<String, String> params){...
    }
}
```

Figura 10 - Estrutura da classe ApiRest.

3.2 Heroku

A Heroku é uma PaaS que corre aplicações desenvolvidas em *Clojure*, *Go*, *Java*, *Node.js*, *PHP*, *Python*, *Ruby* e *Scala*. Utiliza o *Git* como ferramenta de controlo de versões e corre as aplicações em *dynos*, um ambiente Linux virtualizado. O *dyno* que recebe pedidos

HTTP, isto é, onde corre a aplicação, chama-se *web dyno*. Existem ainda *worker dynos* que podem ser ligados ao *web dyno*, para executarem tarefas específicas, e *one-off dynos*, temporários que podem executar tarefas administrativas sem estarem ligados ao *web dyno* (Heroku). A Heroku tem um mercado de *add-ons* grátis e pagos, o Elements Marketplace, que podem ser associados à aplicação para expandir as suas funcionalidades ou como utilitários auxiliares (Heroku). O *interface web* da plataforma tem poucas opções disponíveis. Uma aplicação é totalmente controlada a partir da linha de comandos através do Heroku Toolbelt, que permite "criar/alterar o nome de aplicações, correr one-off dynos, fazer backups, configurar add-ons e gerir o estado da aplicação" (Heroku).

Embora a Salesforce já permita o desenvolvimento completo de serviços e páginas *web*, a Dom Digital optou por integrar um servidor Node.js na Heroku pela razão explicada na secção 3.3.1.

3.2.1 *Middleware* Node.js

O Node.js é um *runtime environment* para desenvolvimento de aplicações *web* em JavaScript. Permite a integração de módulos para expansão de funcionalidades e acelerar o desenvolvimento de aplicações através de *frameworks* (Wikipedia). Os módulos podem ser rapidamente instalados através do *package manager* integrado, o *npm*. A Dom Digital optou por utilizar a *framework express* para o desenvolvimento rápido de um *middleware* que ligasse a RESTful API da Salesforce ao serviço de gestão 3scale. A razão por detrás disto encontra-se explicada no tópico 3.3.1. De qualquer forma, este servidor serve apenas para reencaminhar a *query* com o pedido para o *back-end*.

3.3 3scale

A 3scale é uma fornecedora de serviços de gestão para APIs. Foi integrada na Ardina através do APICast Cloud Hosted Gateway (3scale; 3scale), um *proxy* lançado e configurado automaticamente pela 3scale. Para ser executada, basta indicar o endereço onde se encontra alojada a aplicação Heroku. Os seus serviços incluem registo de utilizadores e controlo de acesso, controlo de limites de chamadas, ferramentas analíticas e de monetização, *developer portal*, e a geração de documentação *live*, o ActiveDocs, com a *framework Swagger*. O último é incluído automaticamente no *developer portal* e, para além de documentar os *endpoints* da Ardina, permite realizar chamadas de teste e ver as respostas devolvidas numa página *web*. Esta é automaticamente construída a partir de um ficheiro JSON, que especifica

o domínio da API, os *endpoints* e as operações disponíveis em cada um, permitindo, também a definição de parâmetros a incluir na chamada (3scale). Na Figura 11 mostra-se a página ActiveDocs da Ardina, construída a partir da especificação indicada na Figura 12.

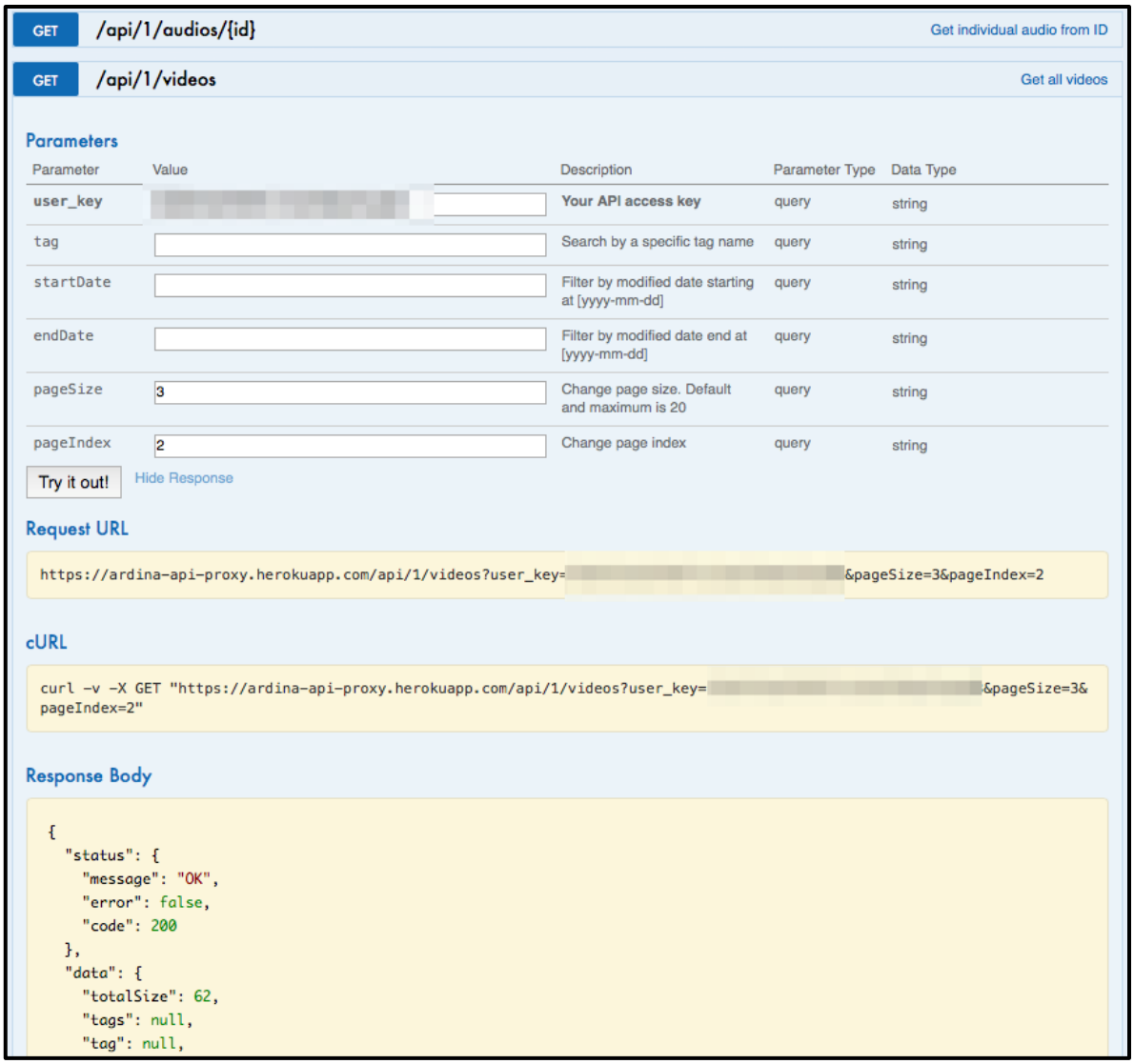

Figura 11 - Página com o ActiveDocs da Ardina.

Figura 12 - Especificação JSON do recurso videos *Ardina, utilizada para construir a página ActiveDocs.*

3.3.1 Gestão Ardina

Embora a Salesforce permita a utilização do OAuth2.0 para autenticação de utilizadores, é necessário que este seja implementado programaticamente. Por isso, a Dom Digital optou pelos serviço da 3scale, que permitem autenticação através da API Key, parametrizada como *user_key*. A integração deste serviço pode ser feita por *proxy* ou *agent*, no entanto, o *proxy* não está preparado para funcionar com Salesforce, nem tão pouco existe *plugin* para tal. Foi, por isso, necessário integrá-los utilizando um servidor com uma tecnologia suportada, tendo sido desenvolvido em Node.js com a *framework express.* Este servidor encontra-se alojado na Heroku e fica entre a plataforma Salesforce e o utilizador final.

3.4 Arquitectura

Na Figura 13 encontra-se a arquitectura da Ardina. Uma aplicação *smartphone/tablet* ou página *web* efectua um pedido que é recebido pelo servidor Node.js. O *proxy* 3scale efectua a autenticação de forma automática, verificando se a *user_key* existe no pedido e se encontra registada no seu sistema. De seguida, o servidor Node.js reencaminha o pedido para

o endereço onde foi exposta a classe ApiRest. Esta processa o pedido, indo buscar os dados à OODBMS, e constrói a resposta apropriada. Antes de chegar à aplicação ou página original, passa outra vez pela camada anterior e o 3scale regista o resultado do pedido.

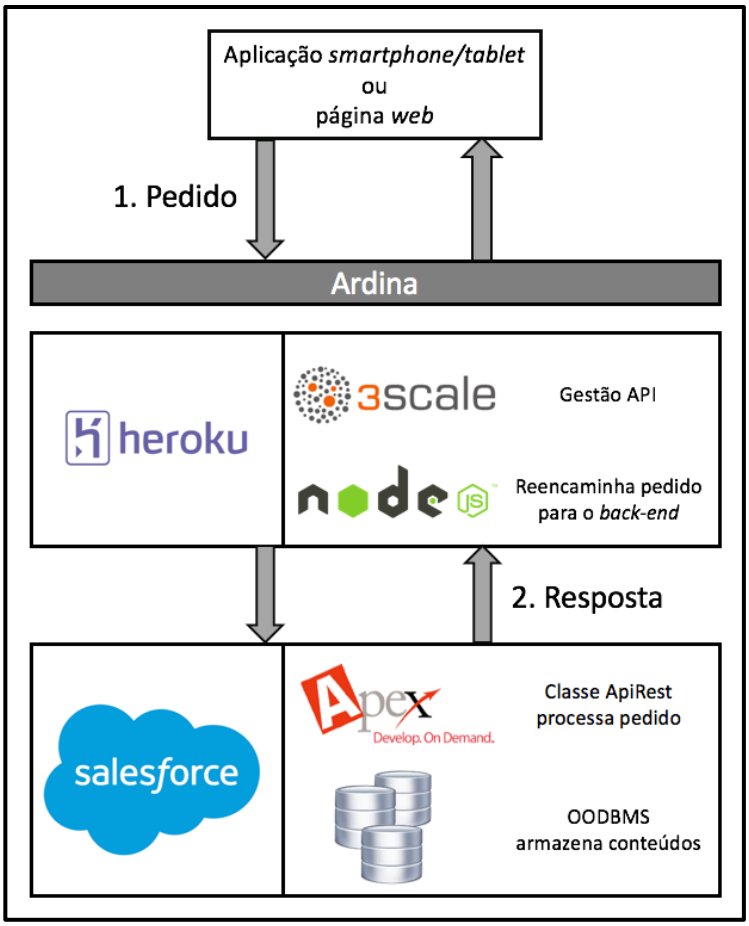

Figura 13 - Arquitectura da Ardina.

4 Metodologia e calendarização

Segue-se a descrição da metodologia de desenvolvimento utilizada durante o estágio e a calendarização das tarefas realizadas.

4.1 Metodologia

A Dom Digital utiliza o Scrum como *framework* para o ciclo de desenvolvimento ágil de *software*. São realizadas reuniões no início da semana para se dar uma visão geral do estado do trabalho e definir os próximos objectivos. Todas as sextas-feiras é entregue um relatório com os resultados das tarefas semanais. Relativamente ao estágio, a atribuição de tarefas e a validação de trabalho eram realizados directamente com a *Chief Technical Officer* (CTO) e o *Chief Executive Officer* (CEO) da Dom Digital. Para a formação, a Dom Digital disponibilizou material didáctico, na forma de cursos *online* e tutoriais das tecnologias utilizadas.

A realização das tarefas foi dificultada pela inexistência de documentação técnica relativa à arquitectura e funcionamento do *software* desenvolvido pela Dom Digital. Para além disso, os comentários no código eram raros e pouco esclarecedores. A situação agravou-se com o facto de dois programadores, um que desenvolveu a aplicação Android RTP Play e outro que implementou a Ardina, já não exercerem funções na empresa. Embora tenha havido tempo para a formação em Apex, a classe ApiRest e a base de dados da Dom Digital são complexas. Para além disso, não se conhecia a tecnologia Node.js e Nginx (utilizada como solução na secção 5.2 e 5.3) o que levou a que muitas tarefas fossem realizadas através de tentativa-erro, pois não havia tempo para adquirir conhecimento e realizar as tarefas propostas. A ajuda era proporcionada pela CTO, que conhecia a arquitectura da Ardina, e por outro colaborador, que implementou algumas funcionalidades no *back-end*. De forma a facilitar desenvolvimentos futuros da Ardina, e embora não tenha sido produzido documentação relevante, todo o código escrito durante o estágio foi comentado.

4.2 Calendarização das tarefas

O mapa de Gantt da Figura 14 mostra a distribuição das tarefas ao longo do estágio.

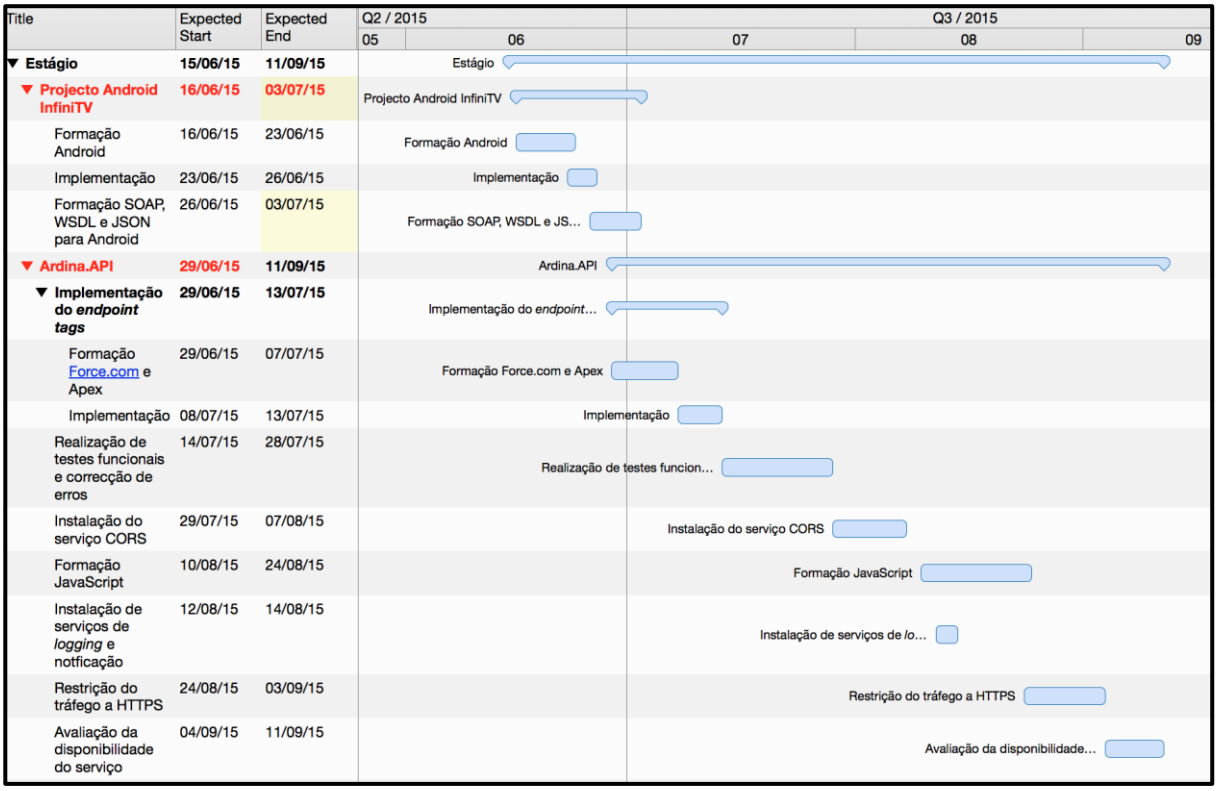

Figura 14 - Mapa de Gantt.

Para experimentar a óptica de programador que desenvolve *software* que consome recursos de uma API, foi proposta a participação no projecto InfiniTV, que consiste no desenvolvimento de uma aplicação Android para visualização de conteúdo *live* e *on-demand* distribuídos pela Minerva (Minerva). O trabalho realizado neste projecto, juntamente com outro colega da Dom Digital, consistiu na adaptação da aplicação Android RTP Play para consumir os conteúdos grátis disponibilizados Minerva, no formato XML. O desenvolvimento desse trabalho, cuja duração foi de 18 dias, não foi incluído no presente relatório, uma vez que não se enquadra directamente no seu tema que é a Ardina.

Atrasos no acesso à API da Minerva levaram a que fosse reencaminhado para o projecto da Ardina. Estava pleanada a realização de testes de *stress* para avaliar a *performance* e a escalabilidade, mencionados na secção 1.2, que acabaram por não ser abordados, tendo sido realizados testes funcionais à Ardina, corrigidos os erros descobertos e instalados os serviços de *logging* e notificação.

5 Expansão da Ardina.API

Neste capítulo encontram-se descritas as tarefas realizadas para expandir as funcionalidades da Ardina, com a implementação de um novo *endpoint*, a devolução de respostas para pedidos provenientes de *browsers* de domínios externos à Ardina, o aumento da segurança com a restrição do tráfego a HTTPS e, por último, a instalação de serviços de *logging* e notificação nos serviços de gestão da API.

5.1 Implementação do *endpoint tags*

Muitos *sites* permitem que o utilizador associe uma ou mais palavras-chave relacionadas com o conteúdo que criam. Estas podem depois ser utilizadas para efeitos de pesquisa e categorização. As *hashtags* do Twitter, com exemplo de uma pesquisa na Figura 15, são, provavelmente, o exemplo mais conhecido (Twitter).

A Dom Digital pretende que a Ardina possua este género de funcionalidades. Para tal, é necessário primeiro aceder às palavras-chave existentes na Ardina. Embora já exista conteúdo que tenha associado uma ou mais palavras-chave, não existia forma directa de se aceder a este atributo. Portanto, propôs-se que fossem desenvolvidos dois novos *endpoints* que permitam:

- A devolução de todas as palavras-chave associadas aos itens (ficheiros áudio, vídeo, imagens e conteúdos).
- A devolução de uma palavra-chave com base na sua chave única de identificação (isto é, o seu ID).

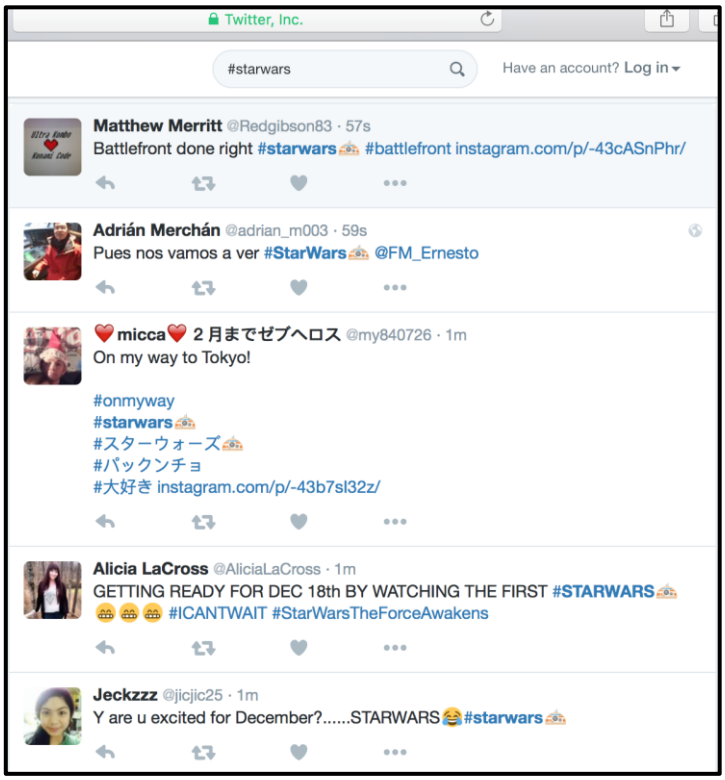

Figura 15 - No Twitter, pode-se pesquisar conteúdo através de hashtags *criadas pelo utilizador.*

A implementação de um novo *endpoint* implica a realização de trabalho em todas as camadas da arquitectura da Ardina:

1. Na plataforma Salesforce, expandiu-se a classe ApiRest para processar os parâmetros dos pedidos para os novos *endpoints*, extrair a informação relevante da base de dados e devolvê-la numa resposta devidamente formatada.
2. Preparou-se o servidor Node.js para receber novos parâmetros.
3. Adicionou-se a documentação necessária dos novos *endpoints* no ActiveDocs.

5.1.1 Expansão da classe ApiRest

Na plataforma Salesforce, expandiu-se a classe ApiRest criando a classe *Tag*, que define os atributos de cada palavra-chave: o seu *ID* e a palavra-chave propriamente dita. Ambos são criados directamente a partir do registo da OODBMS, como mostra a Figura 16.

```
    //eduardo
    //método getTags utiliza classe Tag
    //também tem que ser global
    global class Tag{
        public String id   {get; set;}
        public String name {get; set;}

        public Tag(){}
        public Tag(Tag__c record){
            this.id   = record.Id;
            this.name = record.Name__c;
        }
        public Tag(Tag_Component__c record){
            this.id   = record.Tag__c;
            this.name = record.Tag__r.Name__c;
        }
    } //fim class Tag
}
```

Figura 16 - Classe Tag *define uma palavra-chave.*

No entanto, os atributos de cada *Tag* necessitam primeiro de ser extraídos da base dados, tendo sido desenvolvidos dois métodos auxiliares, indicados na Figura 17, para realizar esta tarefa: *getTags* que extrai todas as palavras-chave existentes para uma lista e o *getSingleTag* que devolve a palavra-chave com o *ID* fornecido na *query*.

```
//eduardo
/**
 * Devolve lista de Tags.
 * @return List<Tag>
 */
global static List<Tag> getTags () {
    List<Tag> tags = new List<Tag>();

    for (Tag__c tag_c : [SELECT Id, Name__c FROM Tag__c]) {
        tags.add(new Tag (tag_c));
    }

    return tags;
}
/**
 * Devolve Tag com Id <reqId>.
 * @return Tag
 */
global static Tag getSingleTag (Id reqId) {
    Tag tag = new Tag([SELECT Id, Name__c FROM Tag__c WHERE Id = :reqId LIMIT 1]);
    if (tag == null) {
        return null;
    } else {
        return tag;
    }
}
```

Figura 17 - Métodos getTags *e* getSingleTag *extraem palavras-chave da OODBMS.*

Desta forma, e tendo em conta a estrutura das respostas como se explica na secção 3.1.3, foram adicionados dois novos construtores à classe *ApiDataClass*, construindo um tipo de resposta para cada caso, isto é, um para uma palavra-chave e outro para a lista de palavras-chave. Por sua vez, isto implicou dois novos construtores na *ApiReturnClass* que contemplem estas respostas.

Por último, foi implementado o processamento dos parâmetros para pedidos que pretendam aceder ao novo *endpoint*, ilustrado na Figura 18. Caso o pedido não contenha um *ID* nos parâmetros, é sinal que o utilizador está a pedir todas as palavras-chave. Caso contrário, a API devolve a palavra-chave encontrada ou dá erro.

```
//eduardo
//novo endpoint
else if (reqPath == 'tags') {
    System.debug('reqPath = ' + reqPath);
    System.debug('req.requestURI = ' + req.requestURI + '|| paramsMap = ' + paramsMap);
    if (reqId != null) { //id da tag
        System.debug('ENTROU NO REQPATH == TAGS COM REQID != NULL');
        Tag thisTag = getSingleTag(reqId);

        //não encontrou tag
        if (thisTag == null) {
            return new ApiReturnClass('Unknown ID: ' + reqId, 400);
        } else {
            return new ApiReturnClass(thisTag);
        }
    } else { //devolve todas as tags
        return new ApiReturnClass(getTags());
    } //end if reqId != null
}
```

Figura 18 - Processamento dos parâmetros para o endpoint tags.

5.1.2 Implementação de novos caminhos no *middleware*

No *middleware*, é preciso definir os *endpoints* e a função que os processa. A Figura 19 mostra os endpoints já implementados na Ardina. Foi necessário definir dois novos caminhos, como mostra a Figura 19: **/tags** é o *endpoint* que devolve todas as palavras-chave e **/tags/:id** devolve a palavra-chave associada ao *ID* fornecido.

```
/* GET home page. */
router.get('/api/'+version+'/images', api.images);
router.get('/api/'+version+'/images/:id', api.images);

router.get('/api/'+version+'/videos', api.videos);
router.get('/api/'+version+'/videos/:id', api.videos);

router.get('/api/'+version+'/audios', api.audios);
router.get('/api/'+version+'/audios/:id', api.audios);

router.get('/api/'+version+'/contents', api.contents);
router.get('/api/'+version+'/contents/:id', api.contents);

//eduardo
//implementação tags
router.get('/api/'+version+'/tags', api.tags);
router.get('/api/'+version+'/tags/:id', api.tags);

router.get('/api/'+version+'/search/:query', api.search);
```

Figura 19 - Caminhos adicionados no middleware *para o novo* endpoint, *delimitados a vermelho.*

Para enviar os pedidos ao *back-end*, o *middleware* tem a função *commonRequest* que aceita como primeiro parâmetro o *endpoint* a aceder. Como tal, foi apenas necessário declarar uma nova função para o *endpoint*, ilustrada na Figura 20.

```
var api = {

  images: function(req, res, next) {
    commonRequest('images',req, res, next);
  },
  videos: function(req, res, next) {
    commonRequest('videos', req, res, next);
  },
  audios: function(req, res, next) {
    commonRequest('audios', req, res, next);
  },
  contents: function(req, res, next) {
    commonRequest('contents', req, res, next);
  },
  //eduardo
  //implementação getTags
  tags: function(req,res,next) {
    commonRequest('tags',req,res,next);
  },
```

Figura 20 - Função api.tags *envia os parâmetros para o novo* endpoint *no* back-end, *delimitados a vermelho.*

5.1.3 Actualização da documentação

Como já se explicou na secção 3.3, a página da documentação e testes é gerada pela *framework* Swagger a partir de um ficheiro JSON onde se encontram definidos os vários *endpoints* da API e os parâmetros que cada um aceita. Como tal, bastou adicionar dois novos *endpoints* na secção própria do 3scale. A Figura 21, mostra a definição do *endpoint* que devolve a palavra-chave com o *ID* fornecido.

Figura 21 - Interface *no 3scale para alterar o JSON do ActiveDocs*.

5.2 Implementação do serviço CORS

Actualmente, os *browsers* implementam o *Same Origin Policy*, uma medida de segurança, que impede que sejam executados *scripts* que acedam a recursos localizados numa origem diferente daquela de onde foi feita o pedido. Isto impede que seja carregado conteúdo perigoso e/ou que o *script* tenha acesso a informação sensível de outra página *web*, prevenindo ataques de *Cross-Site Scripting* (Mozilla Developer Network). Esta medida de segurança vem limitar as funcionalidades da Ardina, já que não é garantido que um *website* que consuma os seus recursos se encontre alojado no mesmo domínio.

Uma das formas de se conseguir efectuar pedidos *cross-domain* é através do JSONP (json-p.org), no entanto, a Salesforce apenas permite, nativamente, *output* de dados em JSON e XML (Salesforce.com). Por isto, optou-se por implementar o serviço *Cross-Origin Resource Sharing* (CORS), um mecanismo de segurança recomendado pelo W3C (W3C). Num pedido CORS simples, o cliente envia o seu domínio no cabeçalho HTTP *Origin*, cujo valor é utilizado pelo servidor para decidir se permite o acesso aos seus recursos, caso tenha

implementada uma *whitelist.* Nos casos em que o conteúdo é de acesso público, é utilizado o *wildcard* * no cabeçalho *Access-Control-Allow-Origin* enviado na resposta (Wikipedia). Existem outros cabeçalhos disponíveis que permitem configurar com mais precisão o acesso aos recursos como, por exemplo, que métodos HTTP são permitidos (W3C).

A decisão de implementar o CORS faz ainda mais sentido se considerarmos que JSONP permite apenas pedidos GET (Wikipedia), o que iria limitar a utilização da Ardina com os restantes métodos (PUT, POST e DELETE).

Como já foi mencionado no tópico 3.3, a integração dos serviços 3scale no servidor Node.js foi realizada através da APIcast Cloud Hosted Gateway. No entanto, a equipa de suporte do 3scale informou que esta solução não permite a utilização do CORS e recomendou a instalação de um *proxy* Nginx que implemente esta funcionalidade. Para estes casos, criaram um tutorial *online* (Delgado) que explica o processo de instalação do *proxy*, assim como a configuração do mesmo (Kalkman). Como foi mencionado no tópico 3.3.1, o acesso aos recursos é controlado através de uma API Key, pelo que se optou por utilizar o *wildcard* * na configuração. Na Figura 22 encontra-se ilustrada a nova arquitectura da Ardina, que inclui o novo *proxy* Nginx. Para além de se ter testado com os *sites* em desenvolvimento na Dom Digital, utilizou-se, também, a ferramenta *online test-cors.org* (disponível em http://client.cors-api.appspot.com/client). No pedido de teste, foi devolvido o código 200, como se pode verificar na Figura 23, o que significa que o *proxy* tem o serviço CORS em funcionamento.

Figura 22 - Arquitectura da Ardina após instalação do proxy Nginx.

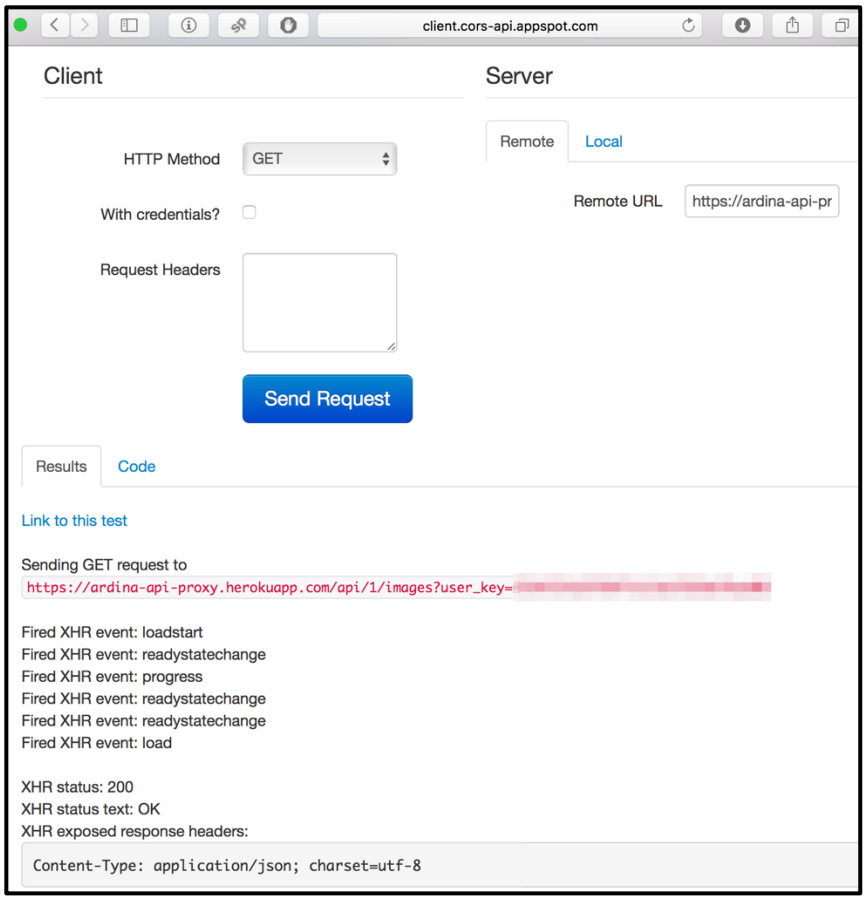

Figura 23 - Ferramenta online *para testar a disponibilidade do serviço CORS num servidor.*

5.3 Restrição do tráfego a HTTPS

Nenhum sistema informático é invulnerável a ataques. O disfarce de um serviço malicioso como legítimo, assim como a leitura e manipulação de dados confidenciais trocados entre dois pontos, são um pequeno exemplo das ameaças existentes (OWASP). A avaliação da segurança é feita com base em três factores, confidencialidade, integridade e disponibilidade, denominados de CIA (do inglês *Confidentiality, Integrity, Availability*). Um sistema que contenha as três propriedades pode ser considerado seguro (Chia, 2012):

- Confidencialidade: capacidade de proteger a informação da leitura ilegítima por terceiros. A encriptação dos dados é uma medida utilizada para garantir esta propriedade. Num rede informática, os protocolos SSL e TLS, para além de proporcionarem uma forma de identificar as entidades legítimas que prestam serviços, são largamente utilizados na encriptação de dados numa rede informática.

- Integridade: refere-se à prevenção da modificação ilegítima de dados por terceiros, ou seja, os dados em circulação correspondem à realidade e não foram alterados no percurso percorrido desde o envio até à recepção. Tal como acontece na Confidencialidade, a encriptação é uma medida de segurança aplicada para proporcionar Integridade a um serviço.
- Disponibilidade: capacidade de um sistema fornecer a informação requisitada sempre que necessário. Falhas na disponibilidade de um sistema não estão limitadas a ataques informáticos, estando sujeito a erros de *software* e falhas no *hardware* que, para além do *downtime*, podem levar à perda de dados.

Restringir o tráfego a HTTPS é uma forma de proporcionar as capacidades de Confidencialidade e Integridade à Ardina, utilizando certificados de segurança para garantir a identidade e a encriptação dos dados trocados entre a API e a aplicação que fez o pedido. Tendo em conta as alterações efectuadas na arquitectura (mencionadas no tópico 5.2), o redireccionamento do tráfego foi implementado no *proxy* Nginx, já que todos os pedidos são primeiramente recebidos neste ponto. A Heroku disponibiliza os seus certificados SSL gratuitamente para todas as aplicações que não possuam um domínio personalizado, ou seja, para qualquer aplicação que pertença ao domínio *herokuapp.com* ou *heroku.com* (Kerstiens). No caso da Ardina, encontra-se alojada no primeiro domínio.

Para redireccionar todo o tráfego recebido no *proxy* Nginx para HTTPS, basta colocar o bloco de código ilustrado na Figura 24 no início da configuração do caminho raiz (Burek, 2014). Pretende-se que o *proxy* verifique se a ligação foi efectuada com HTTPS antes de avançar para o serviço CORS, de forma a não causar erros de *redirect looping* no *browser*, como aconteceu em várias situações durante testes (antes desta alteração de configuração). A estrutura de controlo *if* verifica se o protocolo do pedido é diferente de HTTPS e, caso seja verdade, reconstrói a URL recebida afixando o protocolo pretendido. O redireccionamento é efectuado com o código 301, de forma a informar que todas as ligações deverão ser actualizadas (Wikipedia).

```
1.  location / {
2.      if ($http_x_forwarded_proto != 'https') {
3.          rewrite ^ https://$host$request_uri? permanent;
4.      }
5.
6.      /* Configuração CORS */
7.  }
```

Figura 24 - Bloco com a lógica de redireccionamento de tráfego no proxy *Nginx.*

Como se pode verificar na Figura 25, os pedidos são encriptados utilizando os certificados SSL fornecidos pelo Heroku, enquanto que todos os pedidos HTTP são agora acompanhados com uma resposta 301 enviada pelo *proxy* (Figura 26).

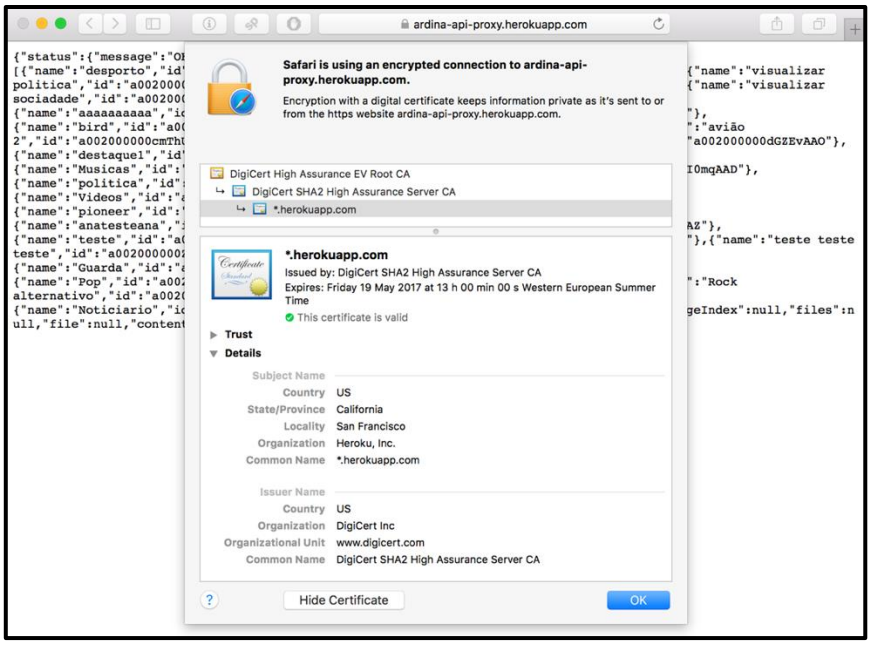

Figura 25 - Ligações efectuadas com a Ardina são automaticamente encriptadas utilizando o certificado Heroku, onde se encontra alojada a aplicação.

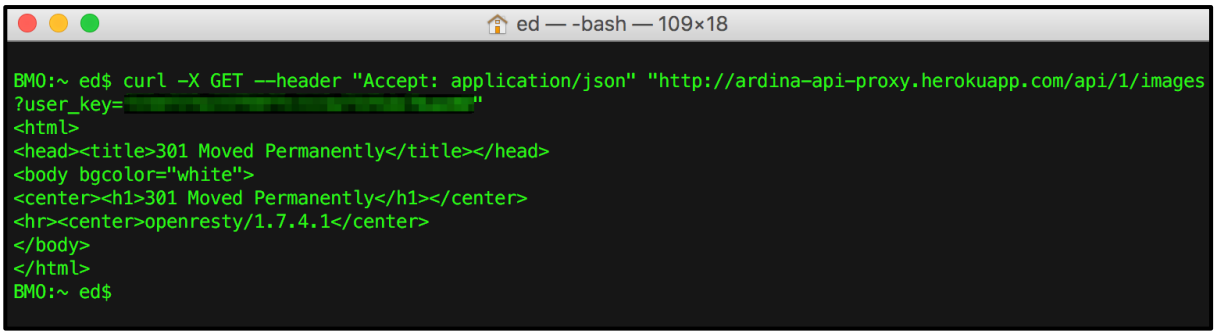

Figura 26 - Código 301 enviado com todos os pedidos HTTP.

Optou-se, também, por encriptar a informação trocada entre o *proxy* Nginx e o servidor Node.js. Para tal, instalou-se o *express-sslify* no servidor Node.js, um módulo que redirecciona automaticamente todo o tráfego HTTP para HTTPS. A configuração é relativamente simples, bastando configurar a instância *express* com o novo módulo, como indicado na linha 3 da Figura 27. O autor do *express-sslify* recomenda a utilização do valor *true* nos casos em que o servidor funciona como *load balancer* (Heinman). No entanto, como só se tem um *web dyno* a funcionar, todos os pedidos chegam sempre ao mesmo.

```
01.  var enforce = require('express-sslify');
02.  var app = express();
03.  app.use(enforce.HTTPS());
```

Figura 27 - Configuração do servidor Node.js *para utilizar o módulo* express-sslify.

5.4 Instalação de serviços de *logging* e notificação

No dia 12 de Agosto, a Ardina recebeu um pico de tráfego, ilustrado na Figura 28, que causou a ultrapassagem dos limites de processamento do *back-end* Salesforce disponibilizados para a Dom Digital, assim como as 61 chamadas/minuto permitidas pelo 3scale.

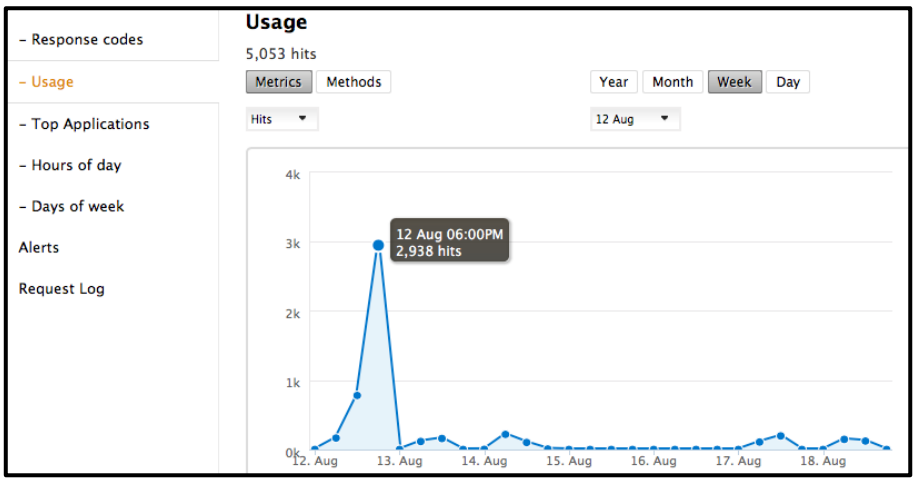

Figura 28 - Tráfego excessivo recebido pela Ardina no dia 12 de Agosto, detectado pelo 3scale.

Determinar a origem do tráfego revelou-se impossível. Os logs mais recentes do servidor Node.js apresentavam apenas informação relativa ao *downtime* da aplicação, relatando que os serviços 3scale se encontram em baixo. Uma vez que a conta Heroku associada permite apenas um log com 300 linhas por aplicação, e esse valor já tinha sido largamente ultrapassado, qualquer informação relativa à origem do tráfego estava irremediavelmente perdida.

Optou-se por instalar o Logentries, um *add-on* da plataforma Heroku que fornece funcionalidades de *logging* e notificações por e-mail para quando sejam detectados certos eventos definidos pelo utilizador (Logentries). A conta grátis permite a configuração de alertas em tempo real para um e-mail personalizável, assim como o armazenamento de 33 MB de *logs* por dia da consola da Ardina. Os eventos são identificados através de padrões

lidos na consola da aplicação, vindo pré-carregado com os erros mais comuns do Heroku (Heroku, 2015), como se pode verificar nos exemplos da Figura 29.

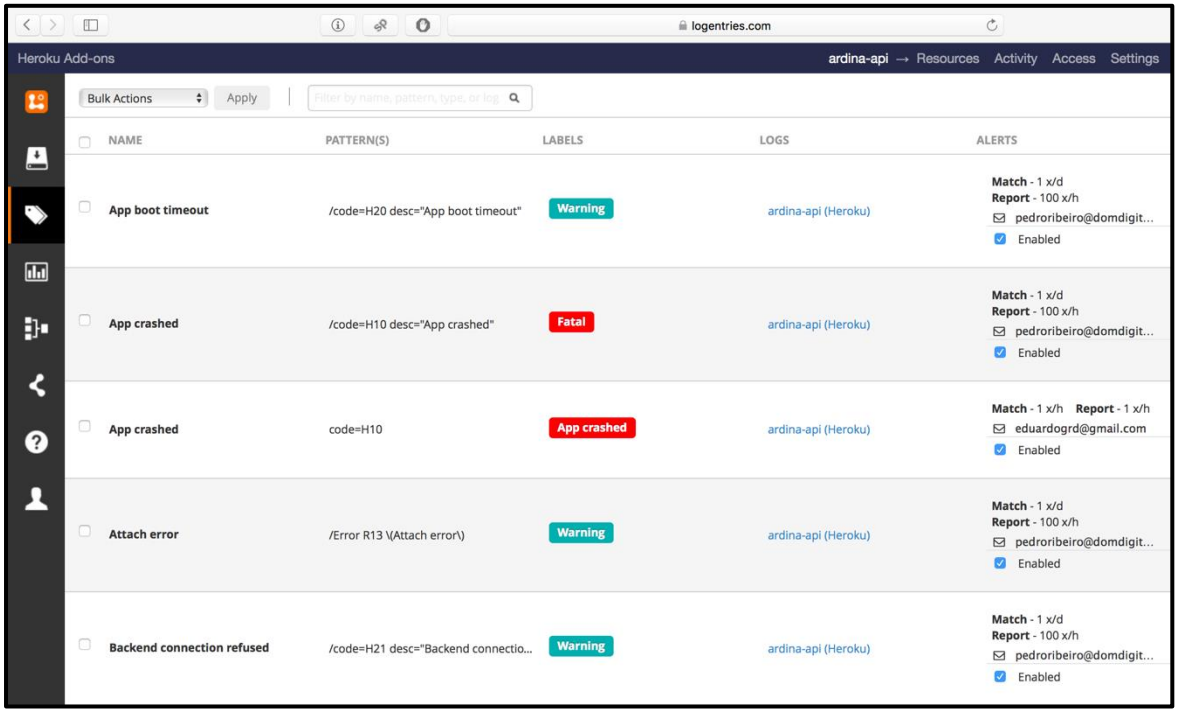

Figura 29 - Erros capturados pelo add-on Logentries da Ardina.

Outro aspecto interessante do *add-on* é a disponibilização de um *interface* para a leitura do *output* através de uma linha temporal, como mostra a Figura 30. Oferece também opções de pesquisa e análise dos resultados, realizando operações de contagem, média, mínimos, máximos, entre outros, para os termos pesquisados nos *logs*.

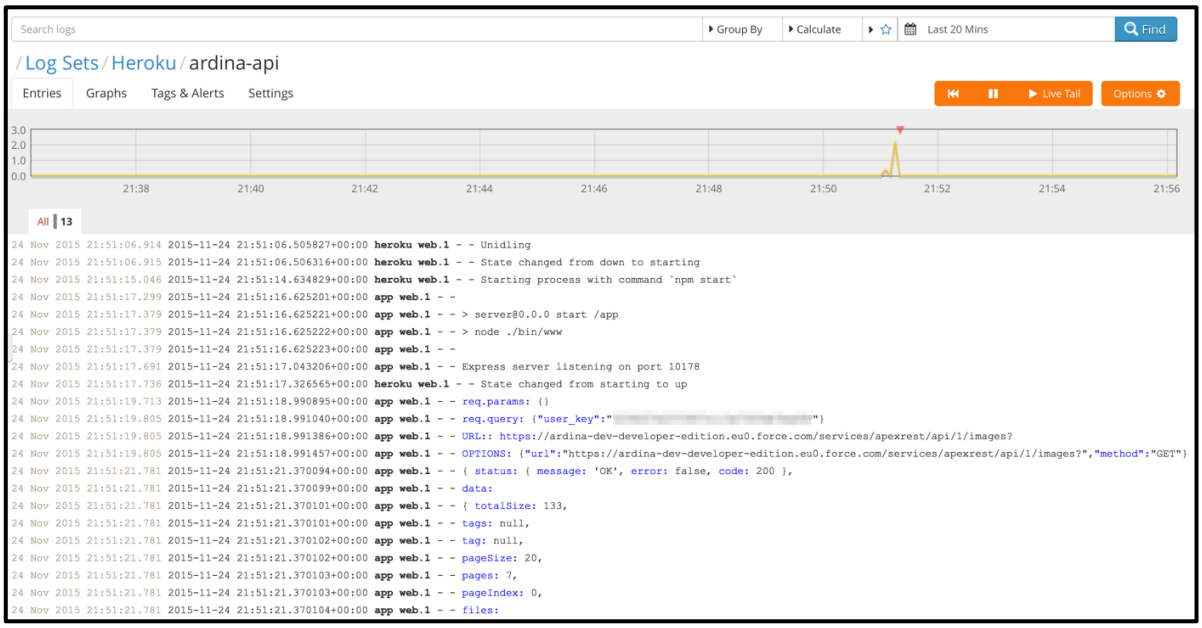

Figura 30 - Logging *da consola da aplicação fornecida pelo Logentries*.

6 Avaliação da Ardina.API

A Ardina foi avaliada com testes funcionais, ou seja, as suas funcionalidades foram testadas consoante o *output* produzido, sem considerar o seu funcionamento interno (Techopedia). Através destes testes foram detectados erros que foram posteriormente corrigidos, melhorando assim a qualidade do produto final. Foi também avaliada a disponibilidade do seu serviço realizando chamadas durante um período de 24 horas. Neste capítulo, são descritos os testes realizados e os procedimentos seguidos para resolver os erros.

6.1 Testes funcionais e correcção de problemas

Foram realizados testes funcionais à Ardina com o objectivo de avaliar o funcionamento correcto de todos os recursos até então implementados. Todo o *input* de teste foi introduzido através do ActiveDocs, que também serviu para verificar o *output* produzido. Para efeitos de *debugging*, utilizou-se a Developer Console da Salesforce e a consola do servidor Node.js.

Os testes realizados consistiram em verificar que:

- Devolve dados com todos os parâmetros preenchidos.
- Devolve dados com cada um dos parâmetros fornecidos.
- Funciona como esperado quando são introduzidos caracteres especiais, tais como: “ç”, “?”, “,”, “.”, “(“, “~”, entre outros.
- Não devolve dados a pedidos com uma *user_key* inválida e pedidos sem *user_key*.
- Aceita apenas o formato da data especificada no ActiveDocs.
- A filtragem por datas funciona correctamente, isto é, para o mesmo filtro de datas, devolve o mesmo conjunto de dados numa chamada e numa *query* realizada directamente à base de dados.

Os erros encontrados não se limitaram apenas a um sub-sistema da Ardina, tendo sido encontrados erros de lógica (embora o *software* não deixe de funcionar, produz resultados anormais) e de *run time* (quando o *software* executa uma operação inválida, por exemplo, a divisão por 0) tanto no *back-end* como no *middleware*. Na Tabela 3 e na Tabela 4 descrevem-se os problemas encontrados nos módulos Node.js e Salesforce, respectivamente, e as respectivas soluções implementadas.

Node.js	
Problema	**Solução**
Servidor deixava de funcionar sempre que eram recebidos caracteres especiais, como por exemplo, “ç” e letras com acento.	Aplicação da função encodeURIComponent() que escapa todos os caracteres excepto os alfabéticos, dígitos decimais, "-", "_", ".", "!", "~", "*", " ' ", "(" e ")" *(Mozilla Developer Network).*
Os parâmetros pageSize e pageIndex, quando usados em conjunto, não devolviam resposta, acontecendo o mesmo com os parâmetros startDate e endDate.	Na construção da string a enviar para o back-end, cada parâmetro era avaliado numa condição utilizando o operador ternário if (<condição> ? <resultado se verdade> : <resultado se falso>), resultando na separação incorrecta dos parâmetros. Por razões deseconhecidas, o problema ficou resolvido trocando o operador ternário pela estrutura if comum. Para evitar problemas futuros, efectuaram-se as mesmas alterações para os restantes parâmetros.

Tabela 3 - Problemas encontrados no módulo Node.js e respectivas soluções implementadas.

Salesforce	
Problema	**Solução**
A pesquisa de ficheiros áudio e vídeo por Id não devolve dados.	As respectivas entidades na base de dados da Salesforce são referenciadas incorrectamente pela classe ApiRest. Os objectos na OODBMS chamam-se **Audio** e **Video**, sendo referenciados por **Audios** e **Videos**, respectivamente. As referências foram corrigidas nas chamadas ao método que faz as *queries.*
A filtragem dos resultados por datas (*startDate* e endDate) não funciona como requerido, resultando em discrepâncias entre os dados devolvidos nas *queries* efectuadas directamente na base de dados e os dados devolvidos pela classe ApiRest.	A diferença nos resultados devia-se ao facto da OODBMS colocar as horas, minutos e segundos a 0 quando é fornecida uma data que só tenha ano, mês e dia. Por exemplo, para um objecto que tenha a data como 2015-01-01T10:10:00 e seja pedida a endDate 2015-01-01 no pedido, o objecto não será incluído nos resultados, já que a OODBMS recebe 2015-01-01T00:00:00 e filtra as datas menores que estas. Para isso, procedeu-se à alteração dos operadores utilizados nas *queries* à base de dados que actuam nas datas. Na *startDate*, é subtraído um dia à data fornecida e é utilizado o operador >, em vez de >= e a data recebida. Para o *endDate*, soma-se um dia e utiliza-se o operador <, em vez de <= e a data recebida.

Tabela 4 - Problemas encontrados no módulo Salesforce e respectivas soluções implementadas..

6.2 Avaliação da disponibilidade

Como já foi mencionado no tópico 5.3, a disponibilidade de um serviço refere-se à sua capacidade para responder a pedidos sempre que necessário. Tratando-se de um serviço *web*, pretende-se que o acesso aos recursos da Ardina estejam disponíveis a qualquer hora do dia.

Foi proposto que se avaliasse a disponibilidade da Ardina realizando chamadas durante um período de 24 horas. Existem várias ferramentas que realizam testes de disponibilidade e de *stress* (Wikipedia), no entanto, tendo em conta que o 3scale fornece dados analíticos do tráfego recebido, optou-se por desenhar um teste que consiste na realização de pedidos em intervalos de 30 minutos durante um período de 24 horas, utilizando o *Gnome Schedule.* Escolheram-se intervalos de 30 minutos devido ao facto do *proxy* Nginx se tratar de um *free dyno*, ou seja, é uma aplicação Heroku que se desliga durante 6 horas se se encontrar activa durante 18 horas seguidas (Heroku, 2015). Esperava-se

que o *proxy* fosse desactivado pela Heroku, mas não foi o caso, tendo-se mantido activo durante as 24 horas do teste. Pensa-se que não basta a aplicação estar activa sem receber pedidos e talvez tenha que os receber continuamente durante as 18 horas seguidas.

O *Gnome Schedule* é uma aplicação disponível para Linux que permite o agendamento simples de tarefas recorrentes ou únicas (Hope). Para o teste, foi agendada a execução do comando da Figura 31, que realiza uma chamada à Ardina através do *cURL* e grava a respectiva resposta num ficheiro *log*. A configuração do *Gnome Schedule* é relativamente simples, como mostra a Figura 32, bastando indicar o comando que se pretende executar e a periodicidade. Embora os pedidos sejam registados pelo 3scale, optou-se por gravar também as respostas na máquina de teste, para que se possam comparar os resultados tanto do lado do cliente como do servidor.

```
curl -vs -o -X GET --header "Accept: application/json" "https://ardina-api-proxy.herokuapp.com/
api/1/tags?user_key=[redacted]" >> ~/ardina_uptime/uptime_log.txt 2>&1;
printf "\n---------------------------------------------------------- //
----------------------------------------------------------\n" >> ~/ardina_uptime/uptime_log.txt
```

Figura 31 - Comando executa uma chamada à Ardina e grava a resposta num ficheiro log.

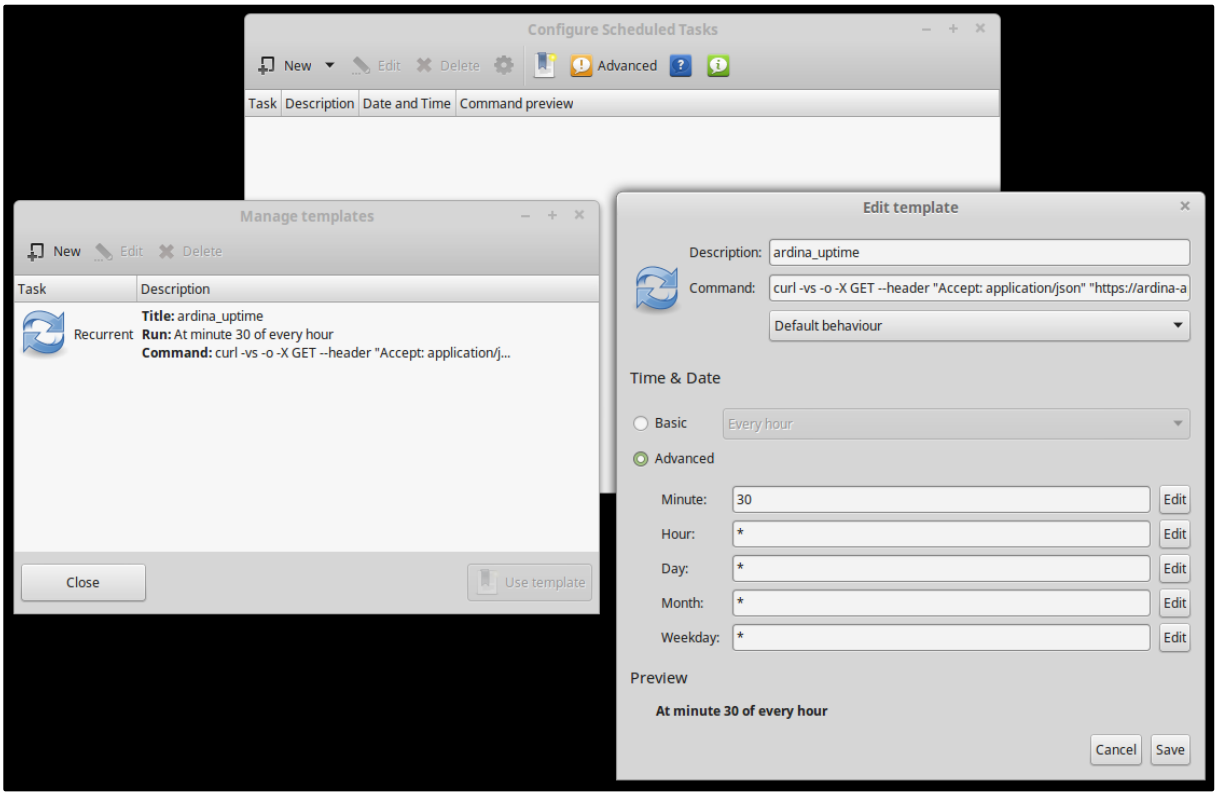

Figura 32 - Agendamento das chamadas à Ardina no Gnome Schedule.

Os testes tiveram início às 20:00 do dia 10 de Setembro e finalizaram às 20:00 do dia 11 de Setembro, tendo sido realizadas 49 chamadas. O 3scale registou-as todas (Figura 33), excepto duas, ocorridas às 01:00 e 18:00 do dia 11, embora tenham sido registadas no *log do Gnome Schedule*. Estas ocorrências podem ser consultadas no Anexo A. O 3scale não registou nenhuma resposta com códigos 4XX (Figura 34) nem 5XX (Figura 35), ou seja, não ocorreram erros do lado do cliente nem do servidor. Na máquina de testes, todas as respostas foram registadas com o código 200, o que significa que a Ardina recebeu o pedido e devolveu os dados em todas as ocasiões. Tirando as duas chamadas mencionadas no parágrafo anterior, os resultados batem certo, tanto do lado do cliente como do servidor. Por estas razões, questiona-se a fiabilidade do 3scale. Durante as 24 horas, a Ardina manteve-se sempre disponível para responder a pedidos.

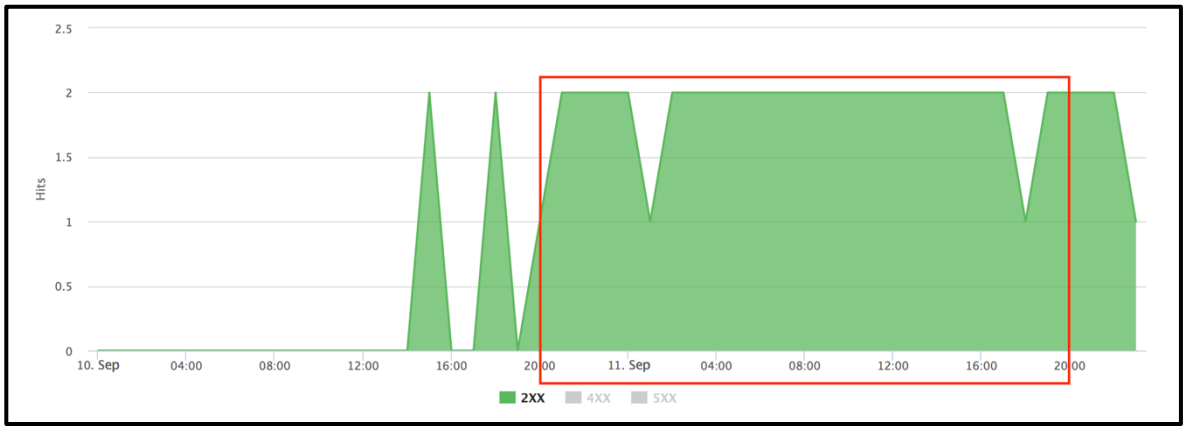

Figura 33 - Respostas com códigos 2XX registadas pelo 3scale. A área delimitada pelo rectângulo vermelho corresponde ao tráfego ocorrido durante os testes.

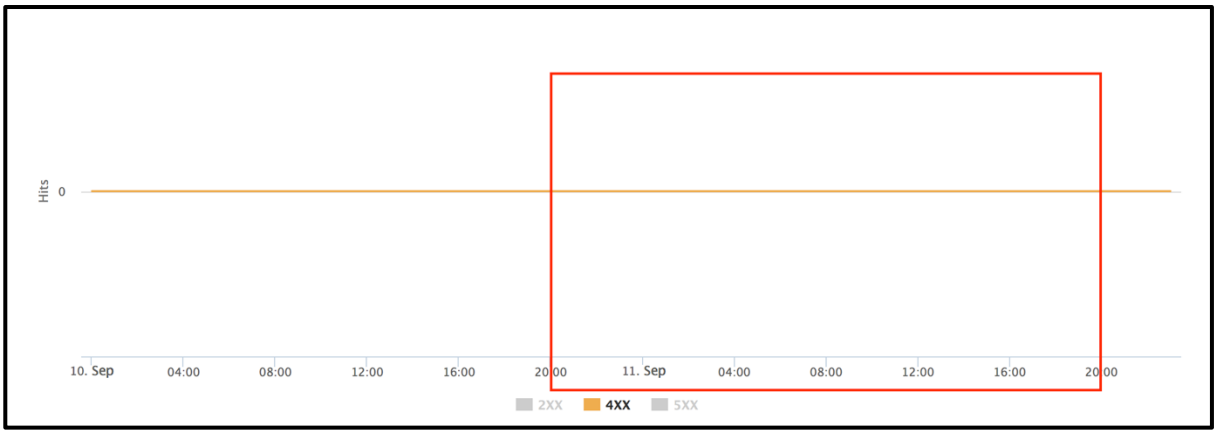

Figura 34 - Respostas com códigos 4XX registadas pelo 3scale. A área delimitada pelo rectângulo vermelho corresponde ao tráfego ocorrido durante os testes.

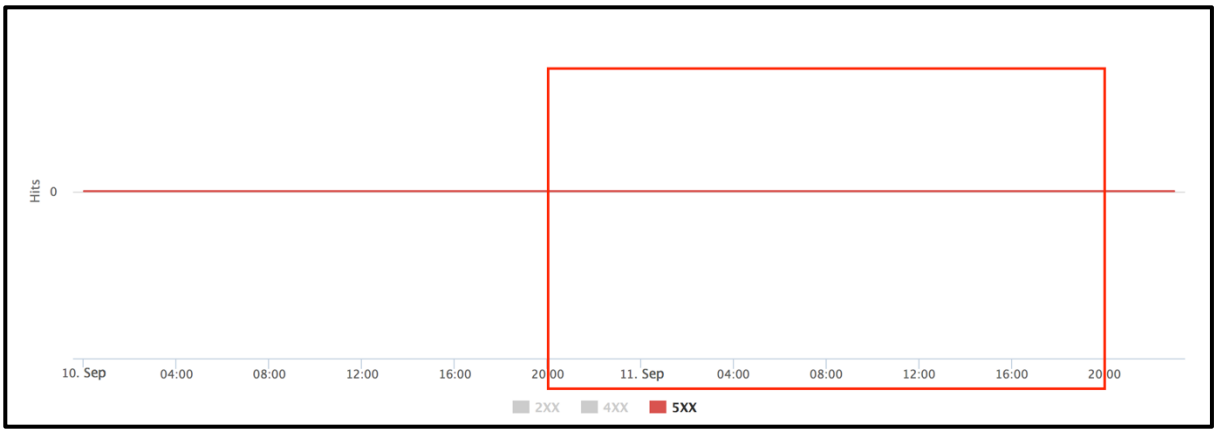

Figura 35 - Respostas com códigos 5XX registadas pelo 3scale. A área delimitada pelo rectângulo vermelho corresponde ao tráfego ocorrido durante os testes.

7 Conclusões e trabalho futuro

7.1 Conclusões

Foram abordados quase todos os conceitos propostos com o trabalho realizado na Ardina durante o estágio. Embora tenha contribuido para o desenvolvimento da aplicação Android InfiniTV, não foi exactamente como se pretendia, pois não cheguei a ter acesso à API da Minerva, devido a atrasos por parte desta empresa. Por constrangimentos de tempo e organização do trabalho, não foram abordados os aspectos de *performance* e escalabilidade. No entanto, foram realizadas outras tarefas não contempladas nos objectivos, mas igualmente enriquecedoras, como foi a realização de testes funcionais e a correcção dos problemas encontrados. Os restantes objectivos foram atingidos, com a implementação de novas funcionalidades na Ardina e actividades de gestão realizadas ao longo do estágio.

Embora os conhecimentos adquiridos durante a licenciatura tenham sido fundamentais na realização das tarefas, as maiores dificuldades surgiram com o facto de ter que implementar funcionalidades em *software* já existente e de relativa complexidade, como foi mencionado na secção 4.1.

A experiência adquirida foi de extrema importância a nível pessoal, tanto pelos desafios encontrados no estágio como na aprendizagem de uma tecnologia de elevada importância, para além de ter ficado a conhecer várias ferramentas novas.

7.2 Trabalho futuro

A arquitectura da Ardina poderá ser melhorada com a troca do fornecedor dos serviços de gestão do 3scale para outro que ofereça melhor integração com a plataforma Salesforce. A existência do *middleware* e do *proxy* aumenta a complexidade da arquitectura e, como tal, há mais camadas onde podem ocorrer falhas.

Mantendo a arquitectura actual, o *dyno* do *proxy* deverá ser alterado para um pago, para evitar a sua desactivação caso seja utilizado mais de 18 horas seguidas. A redundância

do serviço pode ser melhorada grandemente, aumentando o número de *dynos* do *middleware* e do *proxy* (Heroku, 2015).

Deveria ser feito um esforço para a produção de documentação técnica da Ardina, especialmente no que toca à base de dados. Tendo em conta a sua complexidade (que também vai aumentando com o desenvolvimento de novo *software*), os novos colaboradores da Dom Digital terão de fazer um esforço extra considerável para se inteirarem convenientemente da sua estrutura e a sua autonomia de trabalho será reduzida, pois irão necessitar de orientação extra.

Bibliografia

- 3scale. (s.d.). *APIcast Cloud Gateway*. Obtido em 01 de 12 de 2015, de 3scale.net: http://www.3scale.net/api-management/apicast/
- 3scale. (s.d.). *How it works*. Obtido em 01 de 12 de 2015, de 3scale: http://www.3scale.net/how-it-works/
- 3scale Networks S.L. (2011). *What is an API? Your guide to the Internet Business (R)evolution.* Obtido em 01 de 12 de 2015, de 3scale.net: http://www.3scale.net/wp-content/uploads/2012/06/What-is-an-API-1.0.pdf
- Amazon. (s.d.). *Amazon API Gateway Product Details*. Obtido em 01 de 12 de 2015, de Amazon Web Services: https://aws.amazon.com/api-gateway/details/
- Apigee. (s.d.). *API Management*. Obtido em 01 de 12 de 2015, de Apigee: http://apigee.com/about/products/api-management
- Apigee. (s.d.). *Definitive Guide to API Management.* Obtido em 01 de 12 de 2015, de Slideshare: http://www.slideshare.net/apigee/definitive-guide-to-api-management
- Arcitura. (s.d.). *REST Constraints.* Obtido em 02 de 12 de 2015, de What is REST?: http://whatisrest.com/rest_constraints/index
- Bassett, L. (2015). *Introduction to JavaScript Object Notation.* Sebastopol, California: O'Reilly.
- Burek, G. (26 de 10 de 2014). *Require HTTPS to your Heroku app.* Obtido em 28 de 11 de 2015, de Greg Burek: http://blog.gregburek.com/2014/10/26/require-ssl-to-your-heroku-app/

- Chia, T. (20 de 08 de 2012). *Confidentiality, Integrity, Availability: The three components of the CIA Triad.* Obtido em 29 de 11 de 2015, de IT Security Community Blog: http://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/
- Delgado, V. (s.d.). *Tutorial: Deploy an Nginx API gateway on Heroku*. Obtido em 26 de 11 de 2015, de 3scale.net: http://www.3scale.net/2015/07/how-to-deploy-an-nginx-api-gateway-on-heroku/
- DuVander, A. (03 de 06 de 2013). *What Programming Language is Most Popular with APIs?* Obtido em 28 de 11 de 2015, de PRogrammableWeb: http://www.programmableweb.com/news/what-programming-language-most-popular-apis/2013/06/03
- Evernote. (s.d.). *Documentation*. Obtido em 27 de 11 de 2015, de Evernote Developers: https://dev.evernote.com/doc/
- Fawcett, J. e. (2012). *Beginning XML*. Indianapolis, Indiana: John wiley & Sons, Inc.
- GitHub. (s.d.). *Libraries*. Obtido em 27 de 11 de 2015, de GitHub Developer: https://developer.github.com/libraries/
- Google Developers. (s.d.). *YouTube Data API*. Obtido em 27 de 11 de 2015, de Google Developers: https://developers.google.com/youtube/v3/libraries
- Heinman, F. (s.d.). *express-sslify*. Obtido em 26 de 11 de 2015, de npm: https://www.npmjs.com/package/express-sslify
- Heroku. (s.d.). *Command Line*. Obtido em 25 de 11 de 2015, de Heroku Dev Center: https://devcenter.heroku.com/categories/command-line
- Heroku. (23 de 10 de 2015). *Dyno Sleeping and App Recharging.* Obtido em 01 de 12 de 2015, de Heroku Dev Center: https://devcenter.heroku.com/articles/dyno-sleeping
- Heroku. (s.d.). *Elements Marketplace*. Obtido em 01 de 12 de 2015, de Heroku Elements: https://elements.heroku.com
- Heroku. (18 de 11 de 2015). *Heroku Error Codes.* Obtido de Heroku Dev Center: https://devcenter.heroku.com/articles/error-codes
- Heroku. (s.d.). *How Heroku Works*. Obtido em 01 de 12 de 2015, de Heroku Dev Center: https://devcenter.heroku.com/articles/how-heroku-works

- Heroku. (09 de 09 de 2015). *Scaling Your Dyno Formation.* Obtido em 01 de 12 de 2015, de Heroku Dev Center: https://devcenter.heroku.com/articles/scaling
- Hope, G. (s.d.). *GNOME Schedule*. Obtido em 27 de 11 de 2015, de GNOME Schedule: http://gnome-schedule.sourceforge.net
- Internet Assined Numbers Authority. (s.d.). *Media Types*. Obtido em 27 de 12 de 2015, de IANA: http://www.iana.org/assignments/media-types/media-types.xhtml
- Interoute. (s.d.). *What is PaaS?* Obtido em 03 de 12 de 2015, de Interoute: http://www.interoute.com/what-paas
- Javascripting. (s.d.). *Javascripting.* Obtido em 27 de 11 de 2015, de Javascripting: https://www.javascripting.com
- json-p.org. (s.d.). *JSON-P: Safer cross-domain Ajax with JSON-P*. Obtido em 23 de 11 de 2015, de JSON-P: http://json-p.org
- Kalkman, M. (s.d.). *Wide-open CORS config for nginx*. Obtido em 25 de 11 de 2015, de Github Gist: https://gist.github.com/michiel/1064640
- Kerstiens, C. (s.d.). *Announcing Better SSL For Your App*. Obtido em 30 de 11 de 2015, de Heroku Blog: https://blog.heroku.com/archives/2012/5/3/announcing_better_ssl_for_your_app
- Kolb, S. e. (s.d.). *PaaS Profiles*. Obtido em 03 de 12 de 2015, de PaaS Profiles: http://www.paasify.it/vendors
- Linthicum, D. (s.d.). *PaaS Market Moves Beyond Deployment and Scaling.* Obtido em 03 de 12 de 2015, de Restlet: http://restlet.com/blog/2014/05/02/paas-market-moves-beyond-deployment-and-scaling/
- Logentries. (s.d.). *Logentries*. Obtido em 27 de 11 de 2015, de Heroku Elements: https://elements.heroku.com/addons/logentries
- Maddox, S. (03 de 05 de 2014). *Api types.* Obtido em 25 de 11 de 2015, de Slideshare: http://www.slideshare.net/sarahmaddox/api-types
- Minerva. (s.d.). *Clients.* Obtido em 04 de 12 de 2015, de Minerva: http://www.minervanetworks.com/clients/
- Mozilla Developer Network. (s.d.). *encodeURIComponent().* Obtido em 24 de 11 de 2015, de Mozilla Developer: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent

- Mozilla Developer Network. (s.d.). *Same-origin policy*. Obtido em 24 de 11 de 2015, de Mozilla Foundation: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- Mulloy, B. (s.d.). *Web API Design.* Obtido em 24 de 11 de 2015, de apigee: https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf
- NordicAPIs. (s.d.). *Rest vs Soap - NordicAPIs infographic comparison.* Obtido em 23 de 11 de 2015, de NordicAPIs: http://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/
- Oracle. (s.d.). *Java™ Platform, Standard Edition 8 API Specification.* Obtido em 27 de 11 de 2015, de Overview (Java Platform SE 8).
- OWASP. (s.d.). *Category: Attack.* Obtido em 23 de 11 de 2015, de OWASP: https://www.owasp.org/index.php/Category:Attack
- ProgrammableWeb. (s.d.). *APIs by Protocol.* Obtido em 26 de 11 de 2015, de ProgrammableWeb: http://www.programmableweb.com/protocol-api
- ProgrammableWeb. (s.d.). *APIs Dashboard.* Obtido em 26 de 11 de 2015, de ProgrammableWeb: http://www.programmableweb.com/apis
- ProgrammableWeb. (s.d.). *Research Center.* Obtido em 26 de 11 de 2015, de ProgrammableWeb: http://www.programmableweb.com/api-research
- Psistakis, G. (23 de 03 de 2015). *API Management tools: How to find the one for you.* Obtido em 01 de 12 de 2015, de Developer Economics: http://www.developereconomics.com/api-management-tools-how-to-find-the-one-for-you/
- Rouse, M. (s.d.). *API management definition*. Obtido em 01 de 12 de 2015, de TechTarget: http://restlet.com/blog/2014/05/02/paas-market-moves-beyond-deployment-and-scaling/
- Salesforce. (s.d.). *Apex Data Manipulation Language.* Obtido em 25 de 11 de 2015, de Salesforce Developers: https://developer.salesforce.com/docs/atlas.en-us.apex_workbook.meta/apex_workbook/apex6_5.htm
- Salesforce. (s.d.). *Cloud apps and platform.* Obtido em 25 de 11 de 2015, de Salesforce: http://www.salesforce.com/products/
- Salesforce. (s.d.). *Developer Workbooks.* Obtido em 25 de 11 de 2015, de Salesforce Developers: https://resources.docs.salesforce.com/sfdc/pdf/forcecom_workbook.pdf

- Salesforce. (s.d.). *PDF Apex Workbook.* Obtido em 25 de 11 de 2015, de Salesforce Developers: https://resources.docs.salesforce.com/sfdc/pdf/apex_workbook.pdf
- Salesforce. (s.d.). *Salesforce Object Query Language (SOQL).* Obtido de Salesforce Developers: https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm
- Salesforce. (s.d.). *Salesforce Object Query Language (SOQL).* Obtido em 25 de 11 de 2015, de Salesforce Developers: https://developer.salesforce.com/docs/atlas.en-us.198.0.soql_sosl.meta/soql_sosl/
- Salesforce. (s.d.). *SOSL Syntax.* Obtido em 25 de 11 de 2015, de Salesforce Developers: https://developer.salesforce.com/docs/atlas.en-us.198.0.soql_sosl.meta/soql_sosl/sforce_api_calls_sosl_syntax.htm
- Salesforce.com. (s.d.). *Webinar: Apex REST API FAQ*. Obtido em 25 de 11 de 2015, de Salesforce Developers: https://developer.salesforce.com/page/Webinar:_Apex_REST_API_FAQ
- Stormpath. (17 de 04 de 2013). *Secure Your REST API... The Right Way*. Obtido em 27 de 11 de 2015, de Stormpath: https://stormpath.com/blog/secure-your-rest-api-right-way/
- Techopedia. (s.d.). *Functional Testing.* Obtido em 04 de 12 de 2015, de Techopedia: https://www.techopedia.com/definition/19509/functional-testing
- Twitter. (s.d.). *Using hashtags on Twitter.* Obtido em 04 de 12 de 2015, de Help Center: https://support.twitter.com/articles/49309
- Vaqqas, M. (s.d.). *RESTful Web Services: A Tutorial*. Obtido em 24 de 11 de 2015, de Dr.Dobb's: http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069
- W3C. (s.d.). *Cross-Origin Resource Sharing.* Obtido em 23 de 11 de 2015, de World Wide Web Consortium (W3C): http://www.w3.org/TR/cors/
- W3C. (s.d.). *Cross-Origin Resource Sharing.* Obtido em 23 de 11 de 2015, de World Wide Web Consortium (W3C): http://www.w3.org/TR/cors/#syntax
- W3Schools. (s.d.). *JSON Tutorial*. Obtido em 24 de 11 de 2015, de W3Schools: http://www.w3schools.com/json/

- Wikipedia. (s.d.). *Cross-origin resource sharing*. Obtido em 23 de 11 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing#CORS_vs_JSONP
- Wikipedia. (s.d.). *Cross-origin resource sharing*. Obtido em 23 de 11 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing#Simple_example
- Wikipedia. (s.d.). *HTTP 301*. Obtido em 23 de 11 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/HTTP_301
- Wikipedia. (s.d.). *Load testing.* Obtido em 27 de 11 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Load_testing#Load_testing_tools
- Wikipedia. (s.d.). *Node.js*. Obtido em 29 de 11 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Node.js
- Wikipedia. (s.d.). *Platform as a service.* Obtido em 02 de 12 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Platform_as_a_service
- Wikipedia. (s.d.). *Representational State Transfer.* Obtido em 02 de 12 de 2015, de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Representational_state_transfer
- World Wide Web Consortium. (s.d.). *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Obtido em 24 de 11 de 2015, de W3C: http://www.w3.org/TR/REC-xml/
- Zazueta, R. (s.d.). *API Data Exchange: XML vs. JSON*. Obtido em 24 de 11 de 2015, de Mashery: http://www.mashery.com/blog/api-data-exchange-xml-vs-json

Anexos

Anexo A

Este anexo contém as duas respostas recebidas pela máquina de testes e que não foram registadas pelo 3scale, como explicado no tópico 6.2. O rectângulo vermelho delimita o cabeçalho HTTP, onde se encontra registada a data e a hora do pacote HTTP.

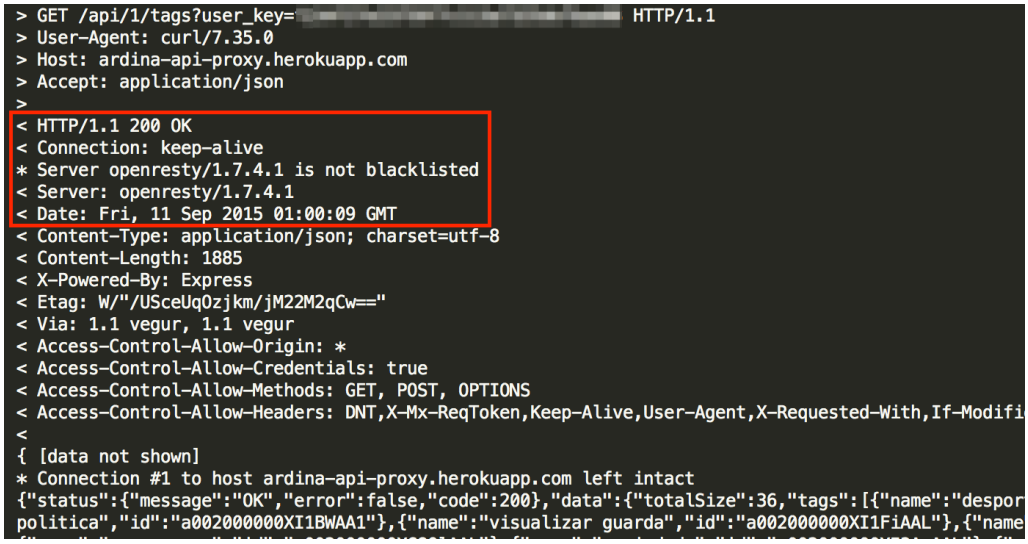

Figura 36 - Resposta recebida pela máquina de testes às 01:00 do dia 11 de Setembro.

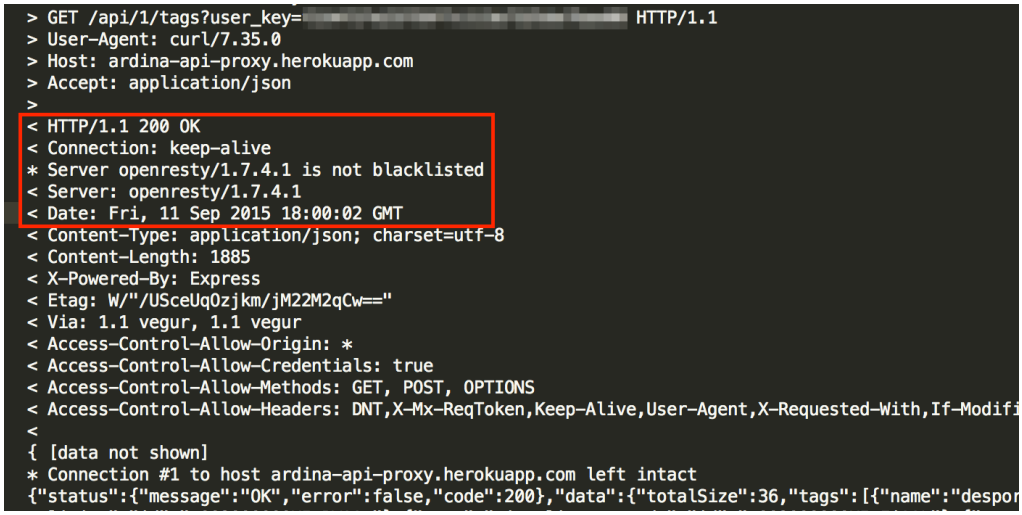

Figura 37 - Resposta recebida pela máquina de testes às 18:00 do dia 11 de Setembro.