INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Arquitetura REST em smartphones Android

PEDRO BRUNO VIVEIROS FERREIRA
Outubro de 2015

POLITÉCNICO
DO PORTO

Arquitetura REST em smartphones Android

Pedro Bruno Viveiros Ferreira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Tecnologias e Conhecimento de Decisão

Orientador: Luís Lino Ferreira

Júri:

Presidente: João Rocha

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, 25 de Outubro de 2015

Página em branco [apagar este comentário]

Dedicatória

Dedico este trabalho aos meus pais, Luís Ferreira e Agostinha Roque de Viveiros, por me terem dado a melhor educação possível, por se terem sempre esforçado para que eu fosse alguém na vida, sem eles não seria capaz de ter terminado este trabalho, eles são uns verdadeiros pilares para mim.

Página em branco [apagar este comentário]

Resumo

As grandes empresas como Google, com o Android, e a Apple, com o iOS, ajudaram a tornar a área das aplicações móveis muito apelativa e obtiveram um elevado sucesso. Com o crescimento elevado nesta área foi necessário usar uma solução que integrasse sistemas e permitisse comunicação entre aplicações diferentes, este é o caso dos Web Services, desta forma as novas aplicações podem comunicar com aplicações já existentes, e permitir que sistemas criados em plataformas diferentes comuniquem.

O objetivo de um Web Service é disponibilizar uma plataforma independente de hardware e também uma plataforma que não implica estar escrita em determinada linguagem de programação. Para o acesso a um Web Service pode ser usado um protocolo ou estilo de arquitetura, nesta tese de mestrado são estudados os protocolos SOAP e a arquitectura REST, que diferem na técnica de acesso, a na sintax das mensagens trocadas. Para determinar qual dos anteriores (REST e SOAP) pode ser o mais adequado a usar no acesso a um Web Service são feitas análises a ambos e comparações entre os tempos de resposta de forma a determinar qual seria o mais vantajoso a nível de performance.

O projeto que serviu como base desta tese de mestrado foi o desenvolvimento de uma aplicação de marcações em serviços beleza/saúde, a qual acede a um Web Service remoto através do REST.

Palavras-chave: Web Services, SOAP, REST, segurança, testes.

Página em branco [apagar este comentário]

Abstract

The creation of applications for smartphones is a recent area, which has had a high growth due to increased adherence by population. Big companies like Google with Android and Apple with iOS helped make this area appealing and with successful. With the high growth in this area arises an necessary to use a solution that integrated systems and enable communication between different applications which is the case of Web Services, this way new applications can communicate with existing applications, and is possible tha systems built on different platforms can comunicate each other.

The purpose of a Web Service is to provide a platform independent of the hardware and also a platform that does not mean to be written in a particular programming language. For access to a Web service can be used a protocol or architecture style, in this work are studied the SOAP protocol and the REST architectural style that differ in their access technique, and in the returned messages. To determine which of the previous (REST and SOAP) may be the most suitable for use in accessing a web service it is made analyzes and comparisons between them, analysing the response times to determine what would be the most advantageous in terms of performance.

The project that was the basis of this thesis was the development of an application of markings on services beauty / health, which accesses a remote Web Service via REST.

Keywords: Web Services, SOAP, REST, security, tests

Página em branco [apagar este comentário]

Agradecimentos

Gostaria de agradecer aos meus pais pelo apoio emocional e pela força que me deram durante o desenvolvimento desta tese de mestrado, e por estarem ao meu lado no momento que mais precisei. Agradeço também aos meus amigos por me terem dado uma força motivacional, principalmente a dois bons amigos que me ajudaram neste processo.

Agradeço também ao orientador Luís Lino Ferreira pela sua disponibilidade e conhecimento que me guiaram na criação da tese de mestrado. Queria também agradecer à empresa Beauti que me deu uma oportunidade de realizar o projeto proposto.

Página em branco [apagar este comentário]

Índice

Inserir página em branco apenas se necessário de modo a que a próxima secção comece numa página à direita

Lista de Figuras

Inserir página em branco apenas se necessário de modo a que a próxima secção comece numa página à direita

Lista de Tabelas

Inserir página em branco apenas se necessário de modo a que a próxima secção comece numa página à direita

Acrónimos e Símbolos

Lista de Acrónimos

SOA	*Service Oriented Architecture* (Arquitetura Orientada a Serviços)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
SMTP	*Simple Mail Transfer Protocol* (Protocolo de Transferência de Correio Simples)
JMS	*Java Message Service*
WWW	*World Wide Web*
MEP	*Message Exchange Pattern* (Padrão de Troca de Mensagens)
REST	*Representational State Transfer* (Transferência do Estado Representativo)
SOAP	*Simple Object Access Protocol* (Protocolo Simples de Acesso a Objetos)
WSDL	*Web Services Description Language*
UDDI	*Universal Description, Discovery, and Integration*
WS-I	*Web Services Interoperability*
WS-Security	*Web Services Security Language*
WADL	*Web Application Description Language*
XML	*eXtensible Markup Language*
JSON	*JavaScript Object Notation*
API	*Application Programing Interface* (Interface de Programação de Aplicações)
BAPI	Beauti API
RPC	Remote Procedure Call (Chamada de procedimento remoto)
MVC	*Model View Controller* (modelo visão controlador)
ASQ	After Scenario Questionnaire (Questionário realizado após a tarefa)

Inserir página em branco apenas se necessário de modo a que a próxima secção comece numa página à direita

1 Introdução

Hoje em dia os smartphones são uma realidade bem presente. Estes dispositivos não só permitem combinar a computação com o telefone, como também oferecem um leque de aspetos inovadores, tais como: fazer *download* de aplicações, capturar imagens, aceder à internet, sensores de localização, etc.

Um dos problemas que um programador se depara em smartphones é o esforço que tem de fazer para desenvolver aplicações para diversas plataformas (Android, iOS, Windows) caso queira abranger um público-alvo maior [Joaquim Anacleto, 2012]. A identificação e análise destes problemas tem levado ao desenvolvimento de várias soluções móveis, entre elas estão os Web Services em que é disponibilizado na internet um serviço que permite a qualquer plataforma a troca de dados.

O aparecimento dos Web Services tem provocado mudanças a nível da integração das aplicações, isto porque com este tipo de serviço existe uma maior otimização de processos. Os Web Services permitem utilizar os benefícios que a internet oferece (jogos online, aplicações multimédia, entre outras) para criar um serviço que permita a que dispositivos distintos troquem dados entre si (comunicar) – mesmo que esses serviços estejam escritos em linguagens (ou sistemas operativos) diferentes.

1.1 Contextualização e Motivação

O projeto desenvolvido para esta tese de mestrado está inserido num contexto empresarial, em que o objetivo é a criação de uma aplicação móvel que faz uso de uma API baseada em REST.

A escolha do tema deveu-se essencialmente ao meu grande interesse pela área de aplicações móveis, nesse sentido foi necessário encontrar uma forma que permitisse troca de dados independente do tipo de plataforma ou linguagem de programação, surgindo então o conceito de Web Service.

Assim, esta tese de mestrado foca-se no uso de tecnologias que permitem a troca de informação estruturada na implementação de Web Services, tais como o protocolo SOAP e a arquitetura REST. É também feita uma análise às tecnologias, SOAP e REST, de forma a obter informação sobre qual a mais adequada a ser utilizada.

As razões pelas quais foram escolhidos o SOAP e o REST para serem analisados foram as seguintes: 1) em relação ao REST é uma alternativa ao SOAP porque é mais leve e simples, a documentação é muito mais fácil de entender, permite diferentes formatos de mensagens como Jason e XML, e tem melhor performance e escalabilidade; 2) em relação ao SOAP este suporta WS-Security que adiciona alguns recursos de segurança; suporta WS-ReliableMessaging que tem uma lógica sucesso/repetição embutido enquanto o REST lida com falhas através de novas tentativas. O SOAP é útil quando necessitamos de criar aplicações com maior exigência de segurança, tais como transações bancárias [Steve Francia, 2012].

1.2 Objetivos

Esta tese tem vários objetivos concretos que são:

- Analisar um Web Service e respetivo funcionamento das tecnologias SOAP e REST;
- Comparar as tecnologias baseadas em SOAP e em REST de modo a determinar qual a mais indicada para usar no acesso a partir de um smarthone;
- Verificar como a aplicação pode trazer valor para os clientes;
- Analisar o funcionamento de uma aplicação baseada em REST, e análise e criação de segurança na aplicação;
- Cumprimento dos requisitos funcionais;
- Analisar o feedback dos utilizadores.

Os objetivos delineados foram atingidos através da execução de algumas tarefas:

- Estudo do protocolo SOAP e da arquitetura REST, onde se pretende perceber as suas características;
- Análise de dados em relação à comparação entre o REST e SOAP;
- Desenvolvimento de uma aplicação baseada na arquitectura REST através de uma API (BAPI – Beauti API) específica para fazer a ligação aos serviços e retornar a informação necessária para o funcionamento da aplicação.
- Estudo da segurança na aplicação baseada em tokens;
- Análise do modelo de negócio;
- Criação de testes de usabilidade de forma a analisar o feedback dos utilizadores.

1.3 Requisitos funcionais

Descrevem explicitamente as funcionalidades e serviços do sistema. Os requisitos funcionais são os seguintes:

- O utilizador poderá fazer login ou se registar na aplicação;
- O utilizador poderá fazer uma pesquisa pelo profissional ou especialidade pretendida;
- A aplicação disponibiliza um ícone que permite ver num mapa onde se situa o profissional de beleza para o qual deseja fazer uma reserva;
- O utilizador poderá escolher o dia e a hora que deseja fazer a sua marcação;
- Cada marcação é mostrada numa outra tela de marcações agendadas;
- O utilizador recebe um email com a sua marcação.

1.4 Estrutura do Documento

Este trabalho encontra-se estruturado em diferentes capítulos, sendo eles a introdução (1), estado de arte (2), comparação entre SOAP vs REST (3), casos de estudo (4), análise da aplicação (5), o modelo de negócio (6), testes de usabilidade (7), e conclusões (6).

O primeiro capítulo serve como introdução e aborda o porquê de ter sido escolhido este tema, quais os objetivos que são pretendidos atingir, qual a metodologia utilizada para atingir esses objetivos, e por fim qual a estrutura do documento.

O segundo capítulo começa por analisar o que é um Web Service e o seu funcionamento, depois são analisados o protocolo SOAP e a arquitetura REST para acesso a um Web Service.

O terceiro capítulo debruça-se comparações realizadas às tecnologias SOAP e REST e sobre alguns testes feitos a pedidos HTTP a servidores baseados em REST ou SOAP, de forma a analisar qual a tecnologia mais adequada a usar na aplicação.

No quarto capítulo são analisados casos de estudo, de forma a analisar outras API's que se baseiam na tecnologia REST usada na aplicação.

O quinto capítulo é dedicado à aplicação, neste são apresentados os requisitos do sistema e do utilizador, a modelação e criação da interface, os diagramas de estados da interface, a segurança na aplicação baseada em tokens, a arquitetura da aplicação mobile cuja é baseada no modelo MVC, a descrição da API baseada na arquitetura REST, foram feitos testes de compatibilidade e usabilidade à aplicação, e algumas conclusões.

O sexto capítulo é dedicado ao modelo de negócio, onde é analisado o produto de forma a compreender como pode criar valor para o público.

No sétimo capítulo são feitos testes de usabilidade que tÊm como finalidade incorporar feedback do utilizador de forma a descobrir a eficácia, eficiência e satisfação do utilizador com a aplicação.

No último capítulo apresentam-se algumas reflexões e as conclusões finais, como também algumas limitações do trabalho e melhorias que podem vir a ser efetuadas.

Inserir página em branco apenas se necessário de modo a que o próximo capítulo comece numa página à direita

2 Estado de Arte

A implementação de um software pode contemplar diferentes linguagens de programação, como Java, C, C++, C#, PHP mas devido às necessidades de integração de sistemas e na comunicação entre aplicações diferentes surge o conceito de SOA (Service Arquitecture Software) com uma solução para esse problema através dos Web Services.

Mas o que são os Web Services? Como se relacionam? Para que serve o REST e o SOAP? Na secção 2 tento responder a estas questões e a outras consideradas importantes para o desenvolvimento do trabalho desta tese de mestrado. É também analisada a segurança de uma aplicação, visto que é algo importante em smartphones.

2.1 Definição de Service Oriented Arquitecture (SOA)

Segundo Bih [Joseph Bih, 2006] uma Arquitetura Orientada aos Serviços (SOA) é uma coleção de serviços que comunicam através de troca simples de dados, ou pode envolver dois ou mais serviços coordenando alguma atividade que precise de meios de conexão entre si.

Para perceber melhor o que é um SOA há que compreender o que é um serviço. Um serviço é uma função bem definida e independente do contexto ou estado de outros serviços. A tecnologia mais comum de conexão de arquiteturas SOA são os Web Services que geralmente usam uma sintaxe baseada em XML.

A figura 3 ilustra a arquitetura básica de um SOA, a qual consiste num produtor de um serviço e num consumidor do serviço. Um consumidor de serviço faz um pedido ao produtor de serviço, o produtor de serviço retorna uma mensagem de resposta ao

consumidor de serviço. As mensagens de pedido e resposta são definidas de uma forma que é compreensível tanto para o consumidor do serviço como para o produtor de serviço. Um produtor do serviço também pode ser um consumidor de um outro serviço.

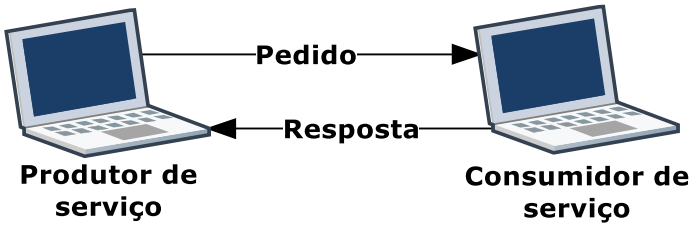


Figura 1 – Arquitetura básica SOA [Joseph Bih, 2006].

2.2 Web Services

Segundo k. Sahin [K. Sahin, 2008], um SOA (Service Oriented Architecture) pode ser implementado em tipos de ambientes de rede diferentes. A implementação de uma arquitetura SOA em ambiente web é denominada de Web Services. O termo Web Services normalmente é associado com serviços que usam o SOAP e os padrões WS* como o WS-Addressing e o WS-Security. A arquitetura REST é também vista como um Web Service quando é aplicada em ambiente web, e refere-se a uma arquitetura baseada em recursos que usa HTTP e XML. O conceito de Web Services é baseado em arquitectura orientada ao serviço, onde uma aplicação pode ser construída através de vários serviços com diferentes funcionalidades.

Um Web Service é uma solução utilizada com o objetivo de realizar a comunicação entre diferentes sistemas, permitindo que a nova tecnologia possa interagir com a tecnologia existente e que sistemas diferentes sejam compatíveis. Um Web Service permite enviar ou receber dados seja em XML ou seja em JSON. Um Web Service é um tipo de API (Application Programming Interface) que trabalha com HTTP, ou SMTP.

Segundo a W3C [W3C, 2004], outros sistemas interagem com o Web Service usando mensagens SOAP ou REST, geralmente transmitidas utilizando HTTP com serialização em XML ou JASON em conjunto com outras normas relacionadas com a Web.

Heather Kreger [Heather Kreger, 2001] afirma que a interface do Web Service esconde os detalhes de implementação do serviço, permitindo que o serviço seja usado

de forma independente da plataforma de hardware ou software no qual é aplicado e também de forma independente da linguagem de programação em que está escrito.

Segundo o site tutorialpoint [tuturialspoint, 2015a], Um web service é um conjunto de protocolos web (HTTP, SMTP, JMS, etc) e padrões abertos (XML, JSON, etc) utilizados para a troca de dados entre aplicações ou sistemas. Um web service pode ser usado por diversas aplicações de software, escritas em qualquer linguagem de programação ou em execução em diferentes plataformas, de forma a possibilitar a troca de dados através de uma rede de computadores como a Internet. Essa interoperabilidade (por exemplo, entre aplicações escritas em Java,ou Python, para Windows ou Linux) é devido ao uso de padrões abertos.

Segundo David Gassner [David Gassner, 2015], um Web Service surge como uma solução para ir buscar conteúdo da internet sem ter-mos de refrescar totalmente a página. Um Web Service é um framework para comunicação entre dois computadores, onde estes comunicam através da internet. Um cliente envia um pedido através da Internet e um servidor recebe esse pedido, processa-o e retorna uma resposta.

A figura 3 ilustra o funcionamento de uma troca de mensagens entre o cliente e o Web Service, em que um cliente envia uma mensagem de pedido a um Web Service, e recebe uma mensagem de resposta.

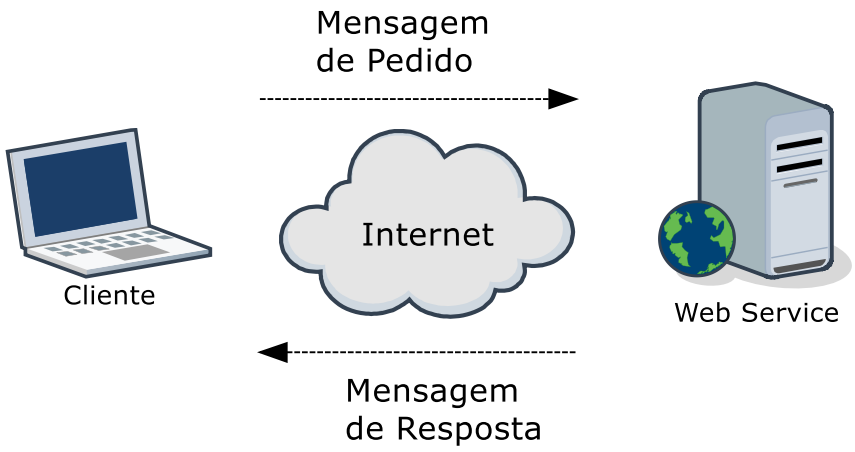


Figura 2 – Conversa entre um cliente e um Web Service [Jakob Jenkov, 2015b].

Jakob Jenkov [Jakob Jenkov, 2015b] afirma que inicialmente o único formato de mensagem utilizado por um Web Service era o SOAP que usa apenas XML, e só apenas mais tarde surgiu o REST que usa HTML, XML e JSON.

Os principais componentes dos Web Services são os seguintes:

WSDL

Quando o Web Service é criado este necessita de ser publicado. Ora um Web Service é independente da linguagem em que foi escrito, para que qualquer plataforma o possa entender.

Para Carla Santos [Carla Santos, 2015], o WSDL descreve as interfaces apresentadas e aponta a localização dos seus Web Services, disponíveis num local da rede, ou seja como aceder aos Web Services de forma confiável. A leitura torna-se fácil e acessível por ser um documento XML.

UDDI

Uma questão que se coloca é como encontrar os Web Services? O UDDI é uma especificação que tem como objetivo descrever, descobrir e integrar Web Services. Segundo Gunzer [GUNZER, 2002] o UDDI é um padrão desenvolvido para fornecer uma diretoria onde se pode guardar detalhes sobre Web Services, que tem como objetivo ser um mediador do serviço, permitindo que os clientes requisitantes encontrem um fornecedor do serviço apropriado.

WS-I

O Web Services Interoperability (WS-I) é um esforço criado para promover a interoperabilidade de Web Services entre plataformas, sistemas operativos e linguagens de programação.

Akash Saurav [Akash Saurav, 2004] afirma que a interoperabilidade refere-se à habilidade para que o *software* e o *hardware* em máquinas diferentes possam comunicar entre si.

2.3 SOAP

Segundo Maurizio Marcese [Maurizio Marchese, 2014], o SOAP ("Simple Object Access Protocol") é um protocolo para transferir dados através da internet. SOAP é um protocolo que se baseia em chamada de procedimento remoto (RPC) e cujas mensagens são representadas em XML, e que se destina à troca de informação estruturada num ambiente distribuído e descentralizado. Uma forma de entender um protocolo é imaginar como se este fosse alguém que está a fazer um pedido em seu nome através de outra entidade.

O SOAP pode ser completamente descrito usando WSDL para descrever a estrutura das mensagens e ações possíveis num endpoint. A tecnologia XML é usada

para definir uma estrutura de mensagens extensível que podem ser trocadas por uma variedade de protocolos subjacentes. A framework é independente de qualquer modelo de programação.

O SOAP é independente do protoloco de transporte, e utiliza o conceito de envelope (header + body). O protocolo SOAP cobre essencialmente quatro áreas:

1) Um **formato de mensagem** para comunicação que descreve como uma mensagem pode ser empacotada num documento XML.

2) Uma **descrição sobre como uma mensagem SOAP** (ou o documento XML que constroi uma mensagem SOAP) deve ser transportada utilizando protocolos diferentes: HTTP (interação através da internet) e SMTP (interação através de email).

3) Um **conjunto de regras** que devem ser seguidas quando uma mensagem SOAP é processada.

4) Um **conjunto de convenções** sobre como transformar uma chamada RPC (Chamada de Procedimento Remoto, ou "Remote Procedure Call" em inglês) em uma mensagem SOAP e o oposto também, bem como a forma de aplicar o estilo RPC de interação.

Os autores Praveen Macerla e Mike Rozlog [Praveen Macherla, 2012] [Mike Rozlog, 2010] defendem que o protocolo SOAP é uma possível solução quando precisamos de:

1) Realizar um processamento assíncrono – O SOAP 1.2 oferece padrões adicionais que garantem a confiabilidade e segurança da aplicação;

2) Realizar contratos formais – se ambos lados (provedor e o fornecedor) têm de concordar no formato de troca, então o SOAP 1.2 dá especificações rígidas para este tipo de interação;

3) Realizar operações de estado – Se a aplicação precisar de informação contextual e gestão de estado de conversação, então o SOAP 1.2 tem uma especificação adicional na estrutura WS* que suporta estas coisas (segurança, transações, coordenação, etc.).

2.3.1 Mensagens SOAP

Segundo Maurizio Marchese, Steve Bratt e a Microsoft [Maurizio Marchese, 2014] [W3C - Steve Bratt, 2004] [Microsoft, 2014], o SOAP define um formato de

mensagem padrão para comunicação, descrevendo como a informação deve ser empacotada (divisão em pedaços menores) no documento XML.

Uma mensagem SOAP é baseada em XML e contém: i) O Envelope: o recipiente do topo que representa a mensagem; ii) O Header: recipiente opcional para recursos adicionados a uma mensagem SOAP de uma forma descentralizada. O SOAP define atributos para indicar quem deve lidar com um recurso e se o entendimento é opcional ou obrigatório; iii) O Body: recipiente obrigatório para informações destinadas ao recetor da mensagem. O SOAP define um elemento para o corpo para reportar erros.

A figura 7 ilustra a estrutura de uma mensagem SOAP, que é composta por envelope, cabeçalho, corpo, e dados da mensagem.

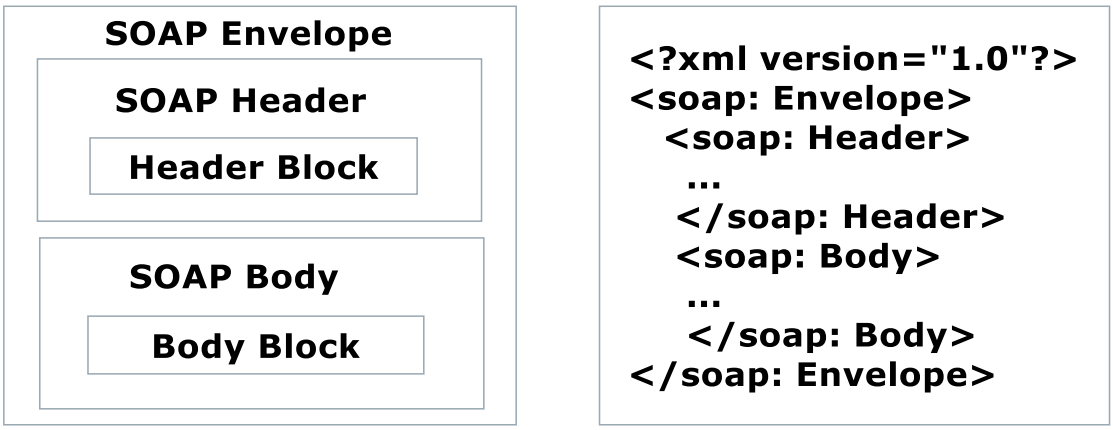


Figura 3 – Estrutura da mensagem SOAP [Ernesto Damiani, 2015].

O código seguinte ilustra um exemplo de uma mensagem SOAP simples.

```
<?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:Header>
    </soap:Header>
    <soap:Body>
        ... message data ...
        <soap:Fault>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Código 1 – Esqueleto de uma mensagem SOAP [w3schools, 2015b].

SOAP Header

Segundo Jakob Jekov [Jakob Jenkov, 2015c], o SOAP Header é um elemento opcional do elemento Envelope, este oferece uma estrutura flexível para a especificação de requisitos adicionais a nível aplicativo.

O elemento Header pode ser utilizado, por exemplo, para especificar uma assinatura digital para serviços protegidos por palavra-passe; de igual modo, ele pode ser usado para especificar um número de conta para serviços de pagamento por uso de SOAP.

O SOAP fornece mecanismos para especificar quem lida com os Headers e o que fazer com eles. O elemento Header pode ter os dois atributos seguintes: 1) **Atributo Actor**: O protocolo SOAP define um caminho de mensagem como uma lista de nós do serviço SOAP. Cada um destes nós intermédios pode executar algum processamento e, em seguida, encaminhar a mensagem para o nó seguinte na cadeia. Ao definir o atributo Actor, o cliente pode especificar o destinatário do SOAP Header; 2) **Atributo MustUnderstand**: Indica se o Header é opcional ou obrigatório. Se for verdadeiro é 1 se for falso é 0. O destinatário tem de entender e processar o atributo cabeçalho de acordo com a sua semântica definida ou retornar uma falha.

O código 2 ilustra como usar o Header na mensagem SOAP.

```
<?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:Header>
 <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
 </m:Trans>
</ soap:Header
...
</soap:Envelope>
```

Código 2 – Exemplo do SOAP Header [w3schools, 2015b].

SOAP Body

O SOAP Body é um elemento obrigatório que contém os dados XML que estão a ser trocados na mensagem SOAP [tuturialspoint, 2015b]. O Body deve ser definido como um elemento filho do envelope e colocado após o Header caso este exista.

O Body contém informação obrigatória destinada para o recetor da mensagem. O código 3 mostra um exemplo.

```
<?xml version="1.0"?>
<soap:Envelope
...
```

```
<soap:Body>
   <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
        <m:Item>Apples</m:Item>
   </m:GetPrice>
</soap:Body>
</soap:Envelope >
```

Código 3 – Exemplo de mensagem para o recetor [w3schools, 2015b].

O exemplo acima solicita a cotação de conjuntos de computadores. Note-se que o m:GetQuotation e os elementos item acima são elementos específicos da aplicação, eles não fazem parte do padrão SOAP.

No código 4 está ilustrada a resposta à query realizada ao código anterior.

```
<?xml version="1.0"?>
<soap:Envelope
........
<soap:Body>
   <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
        <m:Price>1.90</m:Price>
   </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

Código 4 – Exemplo da resposta [w3schools, 2015b].

2.3.2 SOAP – Protocolos de vinculação (binding)

A especificação do SOAP define a estrutura da mensagem e não a forma como esta é trocada [w3schools, 2015a] [W3C, 2007]. Esta lacuna é preenchida pela "vinculação (binding) SOAP", que permitem a que as mensagens SOAP sejam trocadas.

O SOAP não está vinculado a nenhum protocolo de transporte. As mensagens SOAP podem ser trocadas usando os protocolos HTTP ou o SMTP.

O HTTP é síncrono e amplamente utilizado.

O SMTP é assíncrono e é utilizado em última instância ou casos particulares.

SOAP com o HTTP

O cliente identifica o servidor através de um URI, conecta-se a ele usando a rede TCP/IP subjacente, emite uma mensagem de solicitação HTTP e recebe uma mensagem de resposta HTTP pela mesma conexão TCP. O HTTP implicitamente correlaciona a sua mensagem de solicitação com a sua mensagem de resposta.

SOAP com o SMTP

Podemos usar a infra-estrutura do correio electrónico para enviar mensagens SOAP como *emails* de texto ou anexos.

2.3.3 Modelos de comunicação SOAP

Segundo Maurizio Marchese [Maurizio Marchese, 2014] [Michael P. Papazoglou, 2008], os modelos de comunicação SOAP transmitem informações sobre como o conteúdo de um determinado elemento no body ou no header de uma mensagem SOAP é passada entre clientes e servidores.

O SOAP suporta dois tipos de comunicação, a “Remote Procedure Call" (RPC-Style), e a “Document-style” (ou mensagem).

O estilo de comunicação SOAP vem em quatro formas: 1) RPC/Literal; 2) Document/Literal; 3) RPC/Encoded; 4) Document/Encoded.

Modelo de comunicação SOAP: RPC-style

O estilo RPC é uma chamada síncrona de uma operação retornando um resultado. Uma mensagem SOAP contém a pedido e outra mensagem SOAP contém a resposta. As duas aplicações que interagem concordam com a assinatura do método RPC.

```
<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
    <env:Header> … </Header>
     <env:Body>
        <m:GetProductPrice>
               <product-id> 4568 </product-id>
        </m:GetProductPrice>
     </env:Body>
</env:Envelope>
```

Código 5 – Exemplo do modelo de comunicação RPC-style.

Modelo de comunicação SOAP: Document-style

Também conhecido como estilo orientado à mensagem, em que um pedido é feito através do XML. Duas aplicações que interagem entre si necessitam concordar sobre a estrutura dos documentos trocados e só depois usam a mensagem SOAP para transportar esses documentos. O modelo *Document-style* é um estilo de comunicação flexível que proporciona a melhor interoperabilidade utilizando comunicação síncrona ou assíncrona [Thanachart, 2010].

No código 6 está ilustrada a estrutura Document-style, neste caso é enviada toda a informação necessária entre o cliente e o servidor.

```
<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
 <env:Header> … </Header>
 <env:Body>
    <po:PorchaseOrder>
        <po:from>
           <po:accountName> Plastics </po:accountName>
           <po:accountNumber> 123 </po:accountNumber>
        </po:from>
        <po:to>
               <po:supplierName> Plastic Supplies Inc </po:supplierName >
        </po:to>
        <po:product>
               <po:productName> molder </po:productName >
        </po:product>
    </po:PorchaseOrder>
</env:Body>
</env:Envelope>
```

Código 6 – Exemplo do modelo de comunicação Document-style.

2.3.4 O elemento SOAP *Fault*

No SOAP quando um erro ocorre durante o processamento, é retornado no corpo da mensagem o elemento de falha, e a falha é retornada ao remetente da mensagem [tuturialspoint, 2015b] [Jakob Jenkov, 2015a] [Maurizio Marchese, 2014].

O mecanismo de SOAP Fault retorna informações específicas sobre o erro. Tais como um código pré-definido, uma descrição, o endereço do processador SOAP que gerou.

1) Uma mensagem SOAP só pode carregar um bloco de falha; 2) O elemento Fault é uma parte opcional da mensagem SOAP.

```
<?xml version="1.0"?>
<env:Envelope  xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
    <env:Body>
        <env:Fault>

          <env:Code>
            <env:Value>env:Sender</env:Value>
          </env:Code>

          <env:Reason>
            <env:Text xml:lang="en-US">Processing error</env:Text>
            <env:Text xml:lang="da">Processerings-fejl</env:Text>
          </env:Reason>

        </env:Fault>
    </env:Body>
</env:Envelope>
```

Código 7 – Exemplo do elemento SOAP Fault.

2.3.5 O WS-Security

A IBM [IBM, 2002] descreve o WS-Security como uma extensão do SOAP que permite aplicar integridade e confidencialidade ao construir Web Services seguros.

O WS-Security fornece três mecanismos principais: capacidade de enviar tokens de segurança como parte de uma mensagem, a integridade da mensagem, e a confidencialidade da mensagem. Estes mecanismos por si só não proporcionam uma segurança completa, o WS-Security pode ser utilizado em conjunto com outras extensões para fornecer uma maior segurança.

2.4 REST

Segundo Elkstein [Elkstein, 2008], o REST é um estilo de arquitetura que em vez de usar mecanismos complexos como SOAP para ligar máquinas entre si usa o HTTP para fazer chamadas entre máquinas e tem como objetivo fundamentala conceção de aplicações em rede. O REST é uma alternativa mais leve do que o SOAP, e apesar de simples a arquitetura REST apresenta imensos recursos que permitem fazer o mesmo que um Web Service baseado em SOAP e WSDL.

O REST é um estilo de arquitetura em que a instanciação mais comum é REST/HTTP. REST significa Transferência do Estado Representativo ou "Representational State Transfer". Este baseia-se em duas ideias principais: 1) em REST tudo é um recurso; 2) cada recurso tem uma interface uniforme [Paulo Granja, 2014].

Segundo Mike Rozlog e Praveen Macherla [Praveen Macherla, 2012] [Mike Rozlog, 2010], o REST funciona bem para:

1) Largura de banda e recursos limitados: a estrutura de retorno vem em qualquer formato (depende do desenvolvedor). Qualquer *browser* pode ser usado, isto porque o REST usa o padrão GET, POST, PUT, DELETE;

2) Operações totalmente sem-estado: se uma operação precisa ser continuada, então o REST não é a melhor escolha. No entanto se precisamos de operações sem estado CRUD (criar, ler, atualizar, apagar – "Create, Read, Update, Delete"), então o REST é o mais indicado;

3) Situações que exigem cache: se a informação pode ser guardada em cache, então isto é o ideal para a tecnologia, isto porque não é gerada a mesma resposta duas vezes. O caching aumenta a velocidade e reduz o tempo de carregamento de um servidor.

2.4.1 Representações do REST

Os recursos em REST são representados principalmente em XML ou em JSON.

Alexandre Macedo [Alexandre Macedo, 2010] afirma que a vantagem de usar o JSON está no seu formato e interpretação. Em geral o formato JSON costuma ser mais pequeno e mais leve que o XML, o que é uma vantagem quando é necessário que os dados sejam transmitidos através da rede. A principal vantagem do formato JSON está na sua interpretação, por exemplo, no caso das requisições AJAX do javascript.

REST em XML

Segundo Jakob Jenkov [Jakob Jenkov, 2015b], em REST não temos "serviços", mas sim "recursos". Um recurso tem um determinado URL tal como uma página HTML num *web site*. Por exemplo, um exemplo de um recurso pode ser um perfil de utilizador numa aplicação. Tal recurso poderia ter o URL: http://exemplo.com/profiles/pedrobvf.

O código 8 ilustra o pedido REST a um URL

```
GET http://www.example.com/profile.html HTTP/1.1
Host: www.example.com
Accept: application/xml; …
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 …
Accept-Encoding: gzip,deflate,sdch
Accept-Language: pt-PT,en;q=0.8,hi;q=0.6
```

Código 8 – Exemplo de um pedido REST.

O código 9 ilustra o retorno do URL em um documento XML (recurso)

```
<profile>
    <firstName>Pedro</firstName>
    <lastName>Ferreira</lastName>
    <address>
        <street>Caminho do Larano 150</street>
        <zip>9200-121</zip>
        <city>Machico</city>
    </address>
</profile>
```

Código 9 – Exemplo de uma mensagem REST em XML.

REST em JSON

É semelhante ao anterior, exceto que os dados estão representados em JSON. O código 10 ilustra um exemplo do retorno do URL em JSON.

```
{
    firstName : "Pedro",
    lastName  : "Ferreira",
    address   : {
```

```
        street : "Caminho do Larano 150",
        zip    : "12345",
        city   : "Machico"
    }
}
```

Código 10 – Exemplo de uma mensagem REST em JSON.

2.4.2 Mensagens REST

Segundo Vaqqas [M. Vaqqas, 2014], uma mensagem é a comunicação que se dá entre o cliente e o servidor. Por exemplo, o cliente envia uma solicitação ao servidor e este envia uma mensagem de resposta.

Pedido HTTP

Um pedido HTTP tem o formato ilustrado na figura 6.

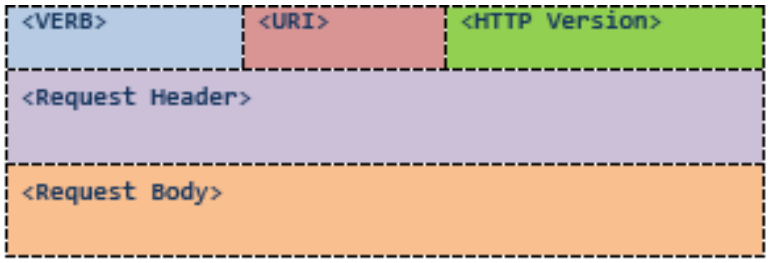


Figura 4 – Formato de um pedido HTTP [M. Vaqqas, 2014].

VERB: é um dos métodos HTTP como o GET, PUT, POST, DELETE, OPTIONS, etc.

URI: é o URI do recurso no qual a operação vai ser realizada

HTTP VERSION: é a versão do HTTP.

Request Header: contém os meta-dados como uma coleção de pares chave-valor de cabeçalhos e seus valores.

Request Body: é o conteúdo da mensagem.

Exemplo de um pedido POST

```
POST http://MyService/Person/
Host: MyService
Content-Type: text/xml; charset=utf-8
Content-Length: 123
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
```

```
  <Country>India</Country>
</Person>
```

Código 11 – Exemplo de um pedido HTTP GET.

Resposta HTTP

Uma resposta REST tem o formato da figura seguinte.

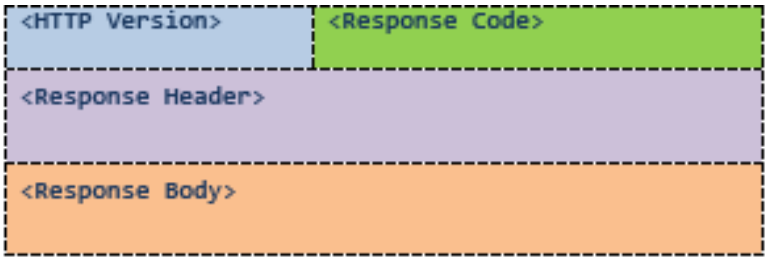


Figura 5 – Formato de uma resposta HTTP [M. Vaqqas, 2014].

Response Code: contém o estado do pedido.

Response Header: contém os meta-dados e as definições sobre a mensagem de resposta.

Response Body: contém a representação se a solicitação foi bem-sucedida.

Exemplo de uma resposta GET

```
HTTP/1.1 200 OK
Date: Sat, 23 Aug 2014 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Sun, 24 Aug 2014 00:31:04 GMT
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    XHTML    1.0    Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>Hypertext Transfer Protocol -- HTTP/1.1</title></head>
<body>
...
```

Código 12 – Exemplo de uma resposta HTTP ao pedido GET.

2.4.3 As três componentes do REST

Segundo a Caelum [Caelum, 2015], uma requisição web tem três componentes, substantivos, verbos e tipos de conteúdo. Isto traz uma vantagem que é conseguir aproveitar toda a estrutura que o protocolo HTTP proporciona.

Substantivos (recursos): são os nomes dos recursos do sistema. Quando fazemos requisições Web precisamos identificar a URI.

Verbos (operações): uma característica do REST é que podemos ter um conjunto pequeno e fixo de operações bem definidas, gerando uma interface uniforme. Deste modo para duas aplicações conversarem elas não precisam implementar diversas operações, basta terem operações definidas. O protocolo HTTP possui sete operações, os métodos GET, POST, PUT, DELETE, HEAD e OPTIONS.

Tipos de conteúdo (representação): os *URIs* devem representar os recursos, as operações no recurso devem ser indicadas pelos métodos HTTP e podemos indicar qual é o formato (HTML, XML ou JSON) em que conversamos com o servidor com o tipo de conteúdo.

2.4.4 Princípios chave da arquitetura REST

Stefan Tilkof [Stefan Tilkov, 2007] [André Gomes, 2008] defende que os cinco princípios fundamentais do REST são: 1) Dar a todos os recurso um identificador; 2) Ligar elementos entre si; 3) Usar métodos padrão; 4) Recursos com múltiplas representações; 5) Comunicação sem estado.

Dar a tosos os recursos um ID

Dar um ID até a coisas não usuais tais como processo, etapa do processo, etc, ou seja, tudo o que identificar deve ter um ID.

Na Web, há o conceito unificado para Ids que é a URI. Utilizar URIs para identificar recursos chave significa ter um ID único e global. Posso usar URIs para identificar tudo o que precisar ser identificado.

Ligar elementos entre si

Todos os recursos que compõem a web podem ser ligados uns aos outros.

A ação do servidor (ou provedor de serviços) de disponibilizar um conjunto de *links* para o cliente (o consumidor do serviço), permite a este último mudar a aplicação de um estado para outro através de uma hiperligação.

Usar métodos padrão

Um navegador Web sabe o que fazer com um URI, porque todos os recursos possuem a mesma interface, ou o mesmo conjunto de métodos (ou operações).

Para que clientes possam interagir com seus recursos, eles devem implementar o protocolo padrão (HTTP) corretamente, isto é, utilizar os quatro métodos também conhecidos como verbos: GET, PUT, POST e DELETE.

Na tabela 1 são apresentados os métodos HTTP usados pelo REST.

Tabela 1 – Métodos do REST [Paulo Granja, 2014].

Método	**Descrição**	**Seguro**	**Indempodente**
GET	Solicita uma representação específica de um recurso.	Sim	Sim
PUT	Cria ou atualiza um recurso com a representação fornecida.	Não	Sim
DELETE	Elimina o recurso especificado.	Não	Sim
POST	Submete dados a ser processados pelo recurso identificado.	Não	Não
HEAD	Semelhante ao GET mas só recupera o Header e não o Body.	Sim	Sim
OPTIONS	Retorna os métodos suportados pelo recurso identificado.	Sim	Sim

Recursos com múltiplas representações

O REST pode ser representado de diversas formas, a tabela 3 ilustra algumas das formas em quem o REST pode ser representado.

Tabela 2 – Possíveis representações do XML.

XML	JSON	YAML
XHTML	Atom	…

Segundo Stefan Tilkof [Stefan Tilkov, 2007], um dos benefícios de ter recursos com múltiplas representações é que um servidor que consume dados em um formato específico não se preocupa com o tipo de específico do cliente, desde que respeite o protocolo da aplicação. Outro benefício de se ter múltiplos formatos é que se for

fornecido um formato HTML e um XML, eles poderão ser consumidos pela aplicação como também por qualquer navegador web, ou seja, a informação ficará disponível para qualquer um.

Comunicação sem estado

Stefan Tilkof [Stefan Tilkov, 2007] defende que, embora o REST inclua a ideia de "não manter estado", não quer dizer que a aplicação não possa ter estados, se tal acontecesse teríamos uma abordagem inútil em muitos cenários. O REST exige que um servidor não guarde o estado da comunicação de qualquer um dos clientes com quem se comunica. A razão mais óbvia este requisito está relacionadas com a necessidade de desenvolver sistemas escaláveis - o número de clientes que podem interagir com o servidor podiam sofrer consideravelmente um impacto se fosse preciso manter o estado do cliente.

2.4.5 Estilo da arquitetura REST

De acordo com o autor John Cowan [John Cowan, 2005] [Rest Api tutorial, 2015] a arquitetura REST é descrita por seis estilos: 1) Interface uniforme; 2) Statless; 3) Cliente-servidor; 4) Em cache; 5) Sistema em camadas; 6) Código sob demanda.

Interface uniforme

Define a arquitetura entre o cliente e o servidor. O que significa que é usada a resposta HTTP, em que o URI representa o nome do recurso, e usamos os métodos HTTP para identificar as ações que a executar sobre esse recurso.

Statless

O servidor não contem nenhum estado do cliente. Isto significa que cada pedido tem contexto suficiente para processar a mensagem, ou seja, cada mensagem é auto descritiva. Qualquer estado da sessão é mantido no lado do cliente.

Arquitetura cliente-servidor

Assume um sistema desconectado, isto é, um cliente nem sempre terá um redireccionamento para uma base de dados. Separação de interesses. A interface uniforme é a ligação entre o cliente e o servidor.

Cacheable

As respostas do servidor são cacheable (o objetivo da cache é não gerar a mesma resposta duas vezes) implicitamente (se não é indicado), explicitamente (o servidor especifica duração, idade máxima, etc.), ou negociadas (o cliente e o servidor negociam para perceber quanto tempo um item ficará em cache).

Sistema em camadas

Uma arquitetura REST pode ter várias camadas de *software*, o que significa que um sistema não pode assumir ligação direta *ao servidor, porque não sabemos ao certo com quem falamos. A arquitetura REST melhora também a escalabilidade.*

Código sob demanda

Um servidor pode temporariamente estender um cliente, transferindo lógica para o cliente. Exemplo disto são os Java applets, ou Javascript executável. É a única restrição opcional.

2.5 Segurança/Confidencialidade

A segurança/confidencialidade é um fator a ter em conta no desenvolvimento de uma aplicação Android, isto porque estes dispositivos apresentam grandes vulnerabilidades que permitem que um utilizador mal-intencionado (hacker) tente roubar os dados pessoais de um outro utilizador.

Uma forma de reforçar a segurança será, por exemplo, colocar a aplicação a comunicar através de um canal seguro como o HTTPS. Poderá ser também usado tokens, ou algoritmos para encriptar a informação do utilizador [Vitor M., 2011], outra forma poderá ser através do uso de tokens.

A encriptação é o processo de transformação da informação inicial em informação ilegível para outros. Este tipo de mecanismo tem o objetivo de enviar a informação confidencial de forma segura, sendo apenas possível a sua descodificação por pessoas autorizadas (que possuam chave de "desencriptação").

A desencriptação é a operação inversa da encriptação. Há dois tipos de criptografia, simétrica e assimétrica.

2.5.1 Criptografia Simétrica

Este tipo de criptografia é conhecido por criptografia de chave secreta. Os algoritmos (DES, 3DES, AES e RC4, etc.) que usam este tipo de criptografia tendem a ser mais rápidos, mas menos seguros que os algoritmos de criptografia assimétrica, isto porque a chave é partilhada entre as várias máquinas.

Funcionamento: Neste tipo de criptografia é usada uma única chave que é partilhada entre o recetor e o emissor, ou seja, a chave usada para cifrar é a mesma para decifrar.

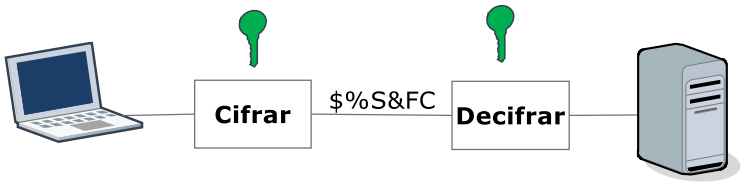


Figura 6 – Criptografia Simétrica [Pedro Pinto, 2010].

2.5.2 Criptografia Assimétrica

Esta criptografia é também conhecida por criptografia de chave pública.

Funcionamento: 1) Utilizam um par de chaves distintas (pública e privada); 2) A chave pública é usada para encriptar (cifrar); 3) A chave privada é usada para desencriptar (decifrar).

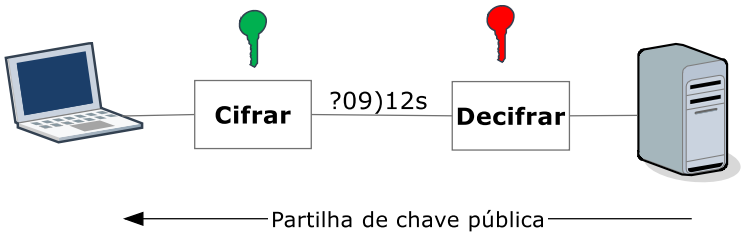


Figura 7 – Criptografia Assimétrica [Pedro Pinto, 2010].

2.5.3 Função de Hashing

Barry Harmsen [Barry Harmsen, 2014] defende que, uma função de hash é um algoritmo que mapeia dados de comprimento variável em dados de comprimento fixo.

Um hash é um número gerado a partir de uma sequência de texto, representado em hexadecimal. As funções que usam criptografia tentam garantir que não é possível voltar à informação original e que o valor de retorno é único, e servem para garantir que um utilizador mal-intencionado consiga descobrir dados confidenciais.

Colisões de Hash

Uma colisão de hash ocorre quando diferentes tipos de input retornam o mesmo resultado, isto é conhecido como colisão de hash.

Na figura 13 está representado uma colisão de hashing. Por exemplo, o Miguel e Manuel têm o mesmo valor de saída, o problema é que existem apenas quarto valores de *output* para cinco valores de *input*, ocorrendo então uma colisão.

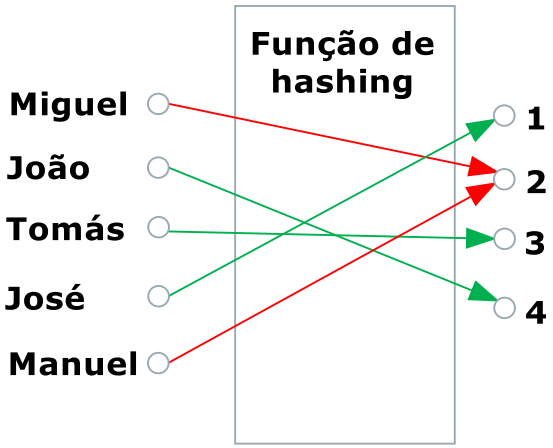


Figura 8 – Colisões de hash

Cálculo da probabilidade de colisão de hash

Para calcular a probabilidade de colisão de hash é aplicada a seguinte formula:

$$1 - e^{\frac{-k(k-1)}{2N}} \quad (1)$$

Resolução de colisão

As duas aproximações de resolução de colisão. A primeira aceita múltiplas colisões, a segunda minimiza as colisões: 1) Encadeamento separado: Colocar as chaves que colidem em uma lista associada a índice; 2) Abrir o endereçamento: Quando uma nova chave colide, encontrar próximo espaço vazio, e colocá-lo lá [Joshua Bloch, 2013].

2.5.4 Implementação de criptografia

Segundo Karan Balkar [Karan Balkar, 2013], se por acaso quisesse implementar a criptografia na aplicação poderia ser utilizada uma técnica de hashing na palavra-chave, a técnica SHA1 ou a MD5.

Em relação ao SHA significa algoritmo de hash seguro. SHA1 produz uma mensagem de 160 bits e é considerado como sendo mais seguro do que o MD5. No entanto, quando comparado com SHA256 o SHA1 não é seguro, porque já foi quebrado.

A função de criptografia MD5 produz um valor hash de 128 bits (16 bytes), e é representado na forma de um número hexadecimal de 32 bits. A desvantagem deste algoritmo é que já foi quebrado.

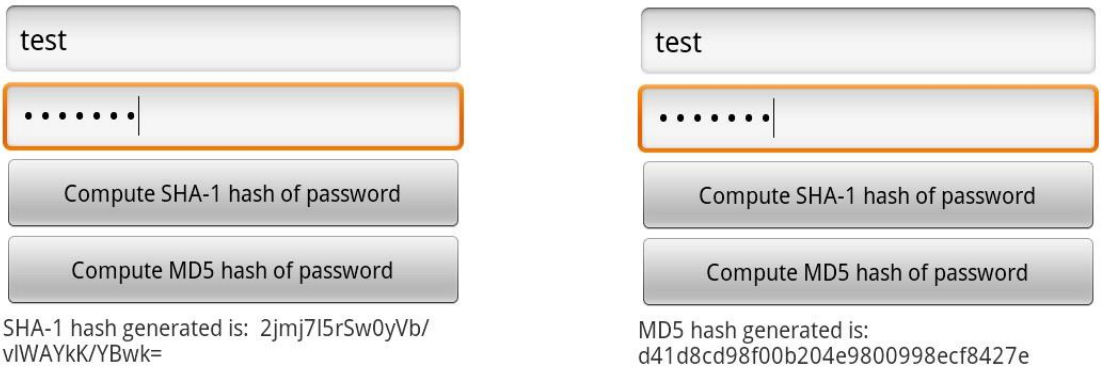


Figura 9 – Exemplo de palavra-chave com hash SHA1 ou MD5 [Karan Balkar, 2013].

A razão pela qual nenhum destes métodos é implementado na aplicação deve-se ao facto de serem métodos inseguros, embora o SHA256 já seja considerado seguro, ainda existe a possibilidade de colisão de hash. No capítulo 5 é abordado o tipo de segurança utilizada na aplicação.

2.6 Conclusões

Criar uma aplicação móvel é algo cheio de desafios, sobretudo se esta for implementada em ambientes de programação diferentes. O conceito de Web Service veio resolver parte do problema ao disponibilizar uma plataforma na web que esteja disponível para qualquer tecnologia poder acedê-la (REST ou SOAP).

Um grande desafio prende-se com o facto da escolha entre a tecnologia (REST ou SOAP) é a mais indicada para aceder a um Web Service, o próximo capítulo ilustra esse tema e determina qual a melhor a usar.

3 Comparação entre REST e SOAP

Este capítulo surge como necessidade de dar resposta a um dos objetivos definidos para esta tese no capítulo 1.2 que é verificar entre o SOAP e o REST qual é o mais indicado a usar na aplicação a ser desenvolvida.

O REST deve ser usado quando clientes e servidores operam num ambiente web, e quando as informações sobre objetos não precisa ser comunicada ao cliente. O REST não deve ser usado quando é necessário realizar transações que envolvem diversas chamadas. Em relação ao SOAP, este deve ser usado quando os clientes precisam aceder a objetos disponíveis no servidor, e quando quisermos cumprir um contrato entre o cliente e o servidor. O SOAP não deve ser usado quando a largura de banda é limitada, e quando quisermos que os programadores usem facilmente a API [Nordic, 2015].

Para tal, em primeiro lugar é feita uma análise entre as tecnologias REST e SOAP. Em segundo são analisados o SOAP e o REST de forma a descobrir o mais vantajoso entre ambos.

3.1 REST vs SOAP

Para perceber melhor o funcionamento do REST e do SOAP convém em primeiro lugar fazer uma breve comparação entre as duas tecnologias e analisar as vantagens e desvantagens de cada uma das tecnologias.

A tabela 3 apresenta algumas comparações entre o REST e o SOAP.

Tabela 3 – Comparação REST vs SOAP

REST	**SOAP**
REST é um estilo de arquitetura [Sonoo Jaiswal, 2014].	SOAP é um protocolo [Sonoo Jaiswal, 2014].
Modelo de comunicação ponto-a-ponto [Jagadeesh Motamarri, 2013].	Desenhado para lidar com computação distribuída, em que a mensagem pode passar entre um ou mais intermediários [Jagadeesh Motamarri, 2013].
É necessário o suporte mínimo de tooling/middleware. Neste caso apenas é necessário suporte HTTP [Jagadeesh Motamarri, 2013].	Requer suporte significante de tooling/middleware [Jagadeesh Motamarri, 2013].
Desenvolvimento mais simples que o SOAP [Kishor Wagh, 2012].	Mais difícil de desenvolver pois requer ferramentas [Kishor Wagh, 2012].
Não há tratamento de erros.	Tratamento de erros incorporado [John Mueller, 2013].
Embutido no modelo de transporte HTTP [Jagadeesh Motamarri, 2013].	SMTP e HTTP são válidas camadas de protocolos utilizadas para transportar o SOAP [Jagadeesh Motamarri, 2013].
Menos detalhado (mais leve) – devido ao uso do JSON [Kishor Wagh, 2012].	Mais detalhado (mais pesado) – devido ao uso de XML [Kishor Wagh, 2012].
As mensagens podem ser em HTML, XML, JSON [Sonoo Jaiswal, 2014].	Apenas utiliza mensagens em XML [Sonoo Jaiswal, 2014].
Usa geralmente mais o GET e o POST para as operações, embora o GET não seja o mais indicado para grandes quantidades de dados. [Mkuchtiak, 2006].	Usa apenas o POST para realizar múltiplas operações [Jean Lima, 2012].
A informação pode ser armazenada em cache.	A informação não pode ser guardada em cache devido ao uso exclusivo do POST.

Através da tabela 3 podemos verificar alguns factos. As mensagens SOAP têm um maior volume de dados (as mensagens estão representadas em xml), em contraste as mensagens REST são o oposto, tornando-as mais adequadas para dispositivos que requeiram menor memória, como o caso dos smartphones. O REST identifica os recursos apenas com um URL, utiliza qualquer um dos verbos HTTP (GET, POST, etc)

para realizar operações, ao contrário do SOAP que apenas usa o HTTP POST. Com o REST podemos armazenar a informação em cache com o HTTP GET, com o SOAP tal não é possível.

3.1.1 SOAP: vantagens e desvantagens

De acordo com a tabela anterior é possível retirar vantagens e desvantagens para o SOAP.

Vantagens

Independente do protocolo de transporte, isto porque uma mensagem SOAP pode ser transportada por HTTP, SMTP, TCP, etc. Os mecanismos de segurança são melhor especificados nos protocolos baseados em SOAP do que no HTTP usado no REST. Os conceitos de transação estão mais bem especificados no SOAP do que no REST.

Desvantagens

Não utiliza todo o poder do HTTP, isto porque usa apenas o POST para realizar múltiplas operações; é muito trabalhoso criar uma interface WSDL embora existam ferramentas para isso; é muito trabalhoso criar a UDDI; O SOAP ao usar o HTTP como mecanismo de transferência é enviado através do pedido POST do HTTP. Apenas os resultados de operações HTTP GET podem ser armazenados em cache, o que significa que não é possível fazer caching ao SOAP devido ao seu uso exclusivo do POST.

3.1.2 REST: vantagens e desvantagens

Vantagens

Interações entre componentes escaláveis; Interfaces gerais; Interação com latência reduzida; Suporta intermediários (proxies e gateways) como transformação de dados e componentes de armazenamento em cache; Separa a implementação do servidor de perceção do cliente de recursos ("URIs recentes não mudam"); Escalável para um grande número de clientes; Permite a transferência de dados em fluxos de tamanho e tipo ilimitado.

Desvantagens

O pedido GET do REST não é indicado para grandes quantidades de dados; ainda não existe um padrão comum aceite para a descrição formal de um serviço REST; o REST não cobre todos os padrões dos Web Services.

3.2 Tecnologia utilizada na análise entre REST e SOAP

Para uma análise mais detalhada às tecnologias REST e SOAP são realizados testes a pedidos HTTP através da ferramenta SoapUI.

Segundo Cristiano Caetano [Cristiano Caetano, 2015], o SoapUI é uma ferramenta open-source cuja principal função é consumir e testar Web Services. O SoapUI suporta todos os protocolos e tecnologias padrão (SOAP e REST). De acordo com este contexto o SoapUI facilita o processo de criação e depuração de testes através uma interface intuitiva e simples de usar, que permite executar testes funcionais.

A escolha desta ferramenta prende-se com a sua facilidade de utilização e outras características. Entre as características destacam-se a importação e geração automática de requisições no WSDL, número ilimitado de requisições por operação, testes funcionais e de carga, execução de diversos testes em paralelo, entre outros.

Para criar um teste ao SOAP no SoapUi é necessário criar um projeto novo através da opção no menu "File -> New SOAP Project" e depois podemos carregar a nossa WSDL, como pode ser visto na figura 14.

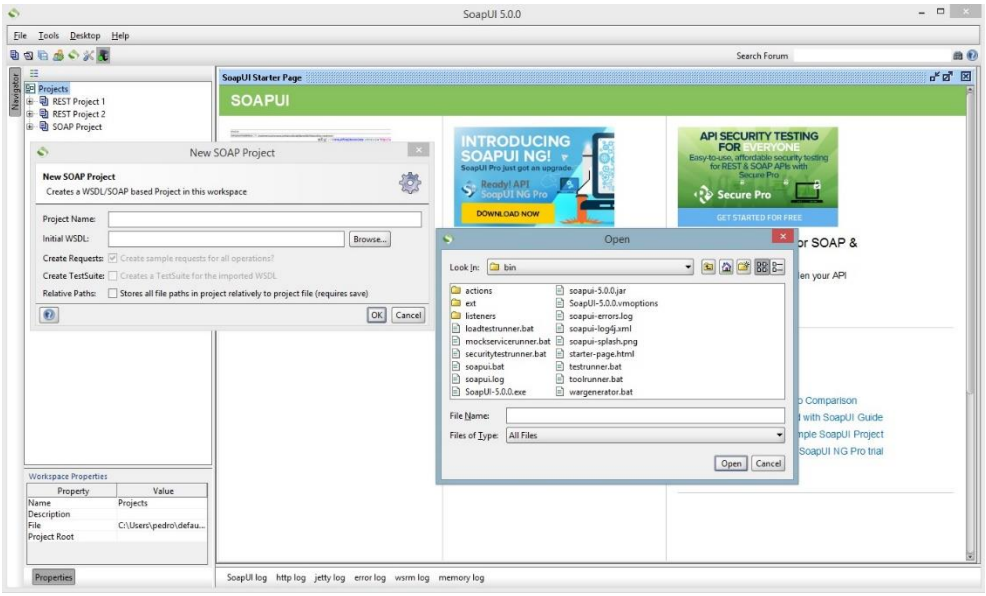


Figura 10 – Criação de um novo projeto SOAP no SoapUI [Cristiano Caetano, 2015].

Para o caso do REST basta ir a "File -> New REST Project" e depois especificar um URI que desejamos analisar, ou então especificar um WADL que não é mais do que um XML com o pedido HTTP.

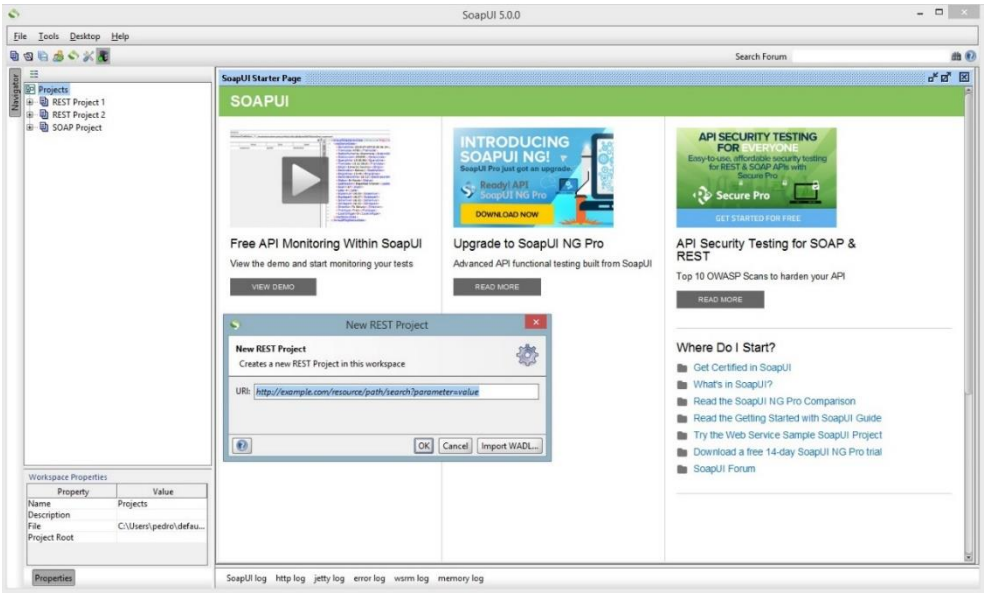


Figura 11 – Criação de um novo projeto REST no SoapUI [Cristiano Caetano, 2015].

3.3 Comparação entre SOAP e REST

Esta seção tem como objetivo testar qual das tecnologias REST e SOAP de acesso a Web Services tem um tempo de resposta mais rápido, sendo que o termo tempo de resposta é compreendido como o tempo que vai desde que submete um pedido até que recebe uma resposta de um específico URL. Para isso foram realizados vários testes a dois Web Services diferentes baseados em SOAP e em REST através de uma rede local sem fios (WLAN) e que permitiu executar os diferentes tipos de testes a partir de Alvalade – Lisboa.

O Web Service baseado em SOAP ilustra uma operação chamada “conversor” cuja finalidade é retornar o resultado de uma conversão de um tipo de moeda para outro, no exemplo seguinte é feita a conversão de florin de aruban para dólar australiano, em que o resultado será de “0.764” [SmartBear Software, 2015a].

Em relação ao Web Service baseado em REST, este ilustra um pedido ao Google Map API, no qual é pedida a referência geográfica para uma determinada morada [SmartBear Software, 2015b].

3.3.1 Análise de submissões ao SOAP e ao REST em XML

Os exemplos apresentados de seguida ilustram testes de submissões a Web Services (que estão em um determinado URL), sendo que cada submissão tem um tempo de resposta diferente, ou seja, é o tempo que o pedido demora a ser executado.

A tabela no anexo A ilustra os resultados entre o REST e do SOAP analisados a partir dos URI's que estão nas fontes da tabela e ilustrados em ecrãs feitos no SoapUI incluídos no anexo A, foram feitos no total 40 submissões aos endpoints URL com pedidos aos Web Services.

Vamos então fazer um agrupamento dos dados de forma a construir os histogramas para analisar a distribuição dos tempos de resposta de cada tecnologia.

A figura seguinte representa o histograma para a tecnologia SOAP, com agrupamento de dados em intervalos de 20 milisegundos.

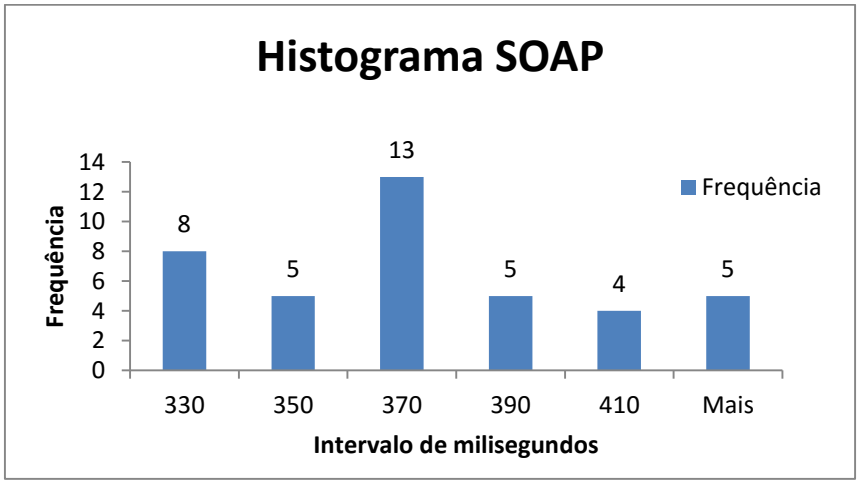


Figura 12 – Histograma: SOAP.

A figura seguinte representa o histograma para a tecnologia REST, com agrupamento de dados em intervalos de 5 milisegundos.

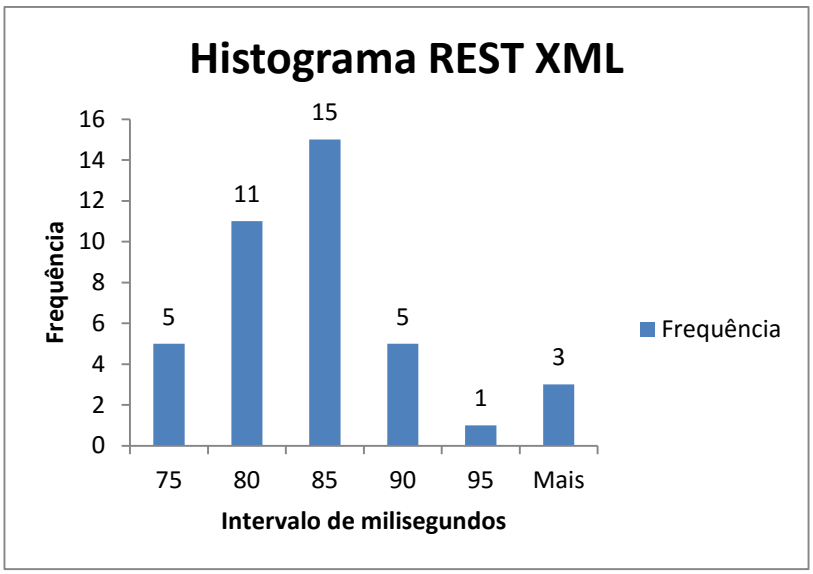


Figura 13 – Histograma: REST em XML.

Os histogramas anteriores seguem uma distribuição normal

Através dos histogramas concluímos que o REST retorna um tempo menor de resposta a um pedido do que o SOAP. Sendo claro que o REST é mais rápido em quase todos os testes.

Análise estatística aos dados do REST e do SOAP

As tabelas seguintes representam uma análise de estatística descritiva feita aos dados representados no anexo A, tendo como objetivo registar as ocorrências do estudo feito e representar os acontecimentos.

Para avaliar os tempos de resposta que o tipo de servidor demora a executar um pedido procedeu-se à análise de uma amostra de 40 submissões feitas a diferentes endpoints URL, correspondentes ao SOAP e ao REST em XML.

Tabela 4 – Análise estatística SOAP.

Análise estatística ao SOAP	
Média	366,275 ms
Mediana	362 ms
Moda	360 ms

Desvio-padrão	34,77914752 ms
Mínimo	317 ms
Máximo	467 ms
Contagem	40

Com o objetivo de avaliar os tempos de resposta que o Web Service demora a submeter um determinado pedido, foi obtida uma amostra na qual constatou-se que maioritariamente o sistema demora 360 ms a responder a um pedido. O Web Service demora no mínimo 317 milisegundos podendo atingir o máximo de 467 milisegundos a responder a um pedido. Dada a grande variedade dos resultados obtidos o desvio padrão para a amostra anterior foi de 34,77 tendo em conta que a média dos resultados ser 366,375. O conjunto de dados apresenta uma mediana de 362.

Tabela 5 – Análise estatística REST em XML

Análise estatística ao REST em XML	
Média	82,825 ms
Mediana	81 ms
Moda	81 ms
Desvio-padrão	14,95615387 ms
Mínimo	45 ms
Máximo	151 ms
Contagem	40

Para esta segunda amostra foi obtida uma mediana de 81 milisegundos. Sendo que o Web Service demora no mínimo 45 milisegundos a responder a um pedido podendo atingir o máximo de 151 milisegundos. O desvio padrão para a segunda amostra foi de 14,95615387, tendo em conta que a média dos resultados é 82,825 ms. O conjunto de dados apresenta uma mediana de 81.

Pela comparação da análise estatística entre o REST e o SOAP conclui-se que o REST obtém valores mais baixos quando toca a efetuar pedidos a um endpoint URL.

3.3.2 Análise de submissões ao REST em XML e ao REST em JSON

Neste capítulo é pretendido descobrir qual o melhor formato de dados a usar na tecnologia REST, se XML ou JSON.

Para fazer estes testes foram comparados os resultados entre o REST em XML e a o REST em JSON, esta comparação encontra-se ilustrada no Anexo B.

A figura seguinte representa o histograma para a tecnologia REST em JASON, com agrupamento de dados em intervalos de 5 milisegundos.

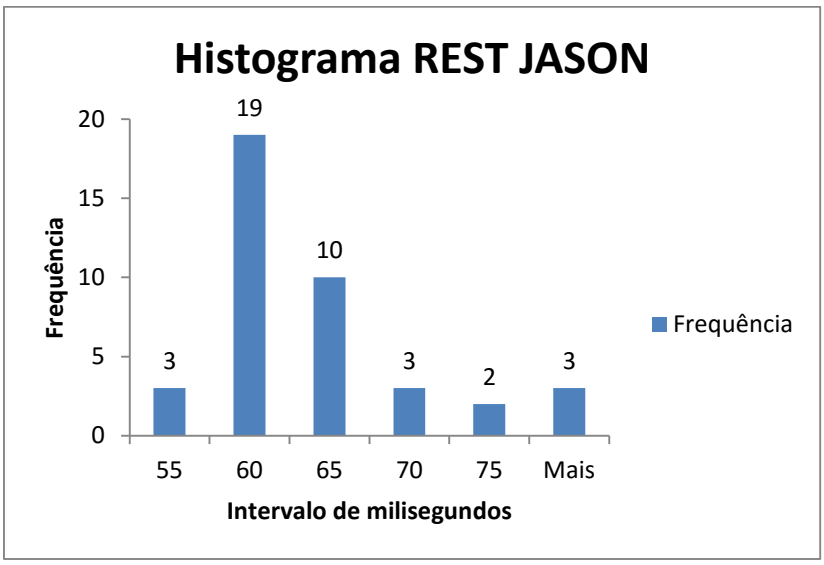


Figura 14 – REST em JSON: Histograma.

Pela comparação entre a figura 13 e a figura podemos concluir que o formato de dados do REST em JSON comparativamente ao formato de dados do REST em XML melhorou os tempos de resposta, que são menores do que no formato XML.

Análise estatística ao REST em Json

Tabela 6 – Análise estatística REST em Json

Análise estatística ao REST em Json	
Média	63,3 ms
Mediana	60 ms
Moda	60 ms
Desvio-padrão	10,32596925 ms
Mínimo	55 ms
Máximo	115 ms
Contagem	40

Nesta última amostra de tamanho 40, foi obtido uma mediana de 60 milisegundos. O Web Service demora no mínimo 55 milisegundos a responder a um pedido podendo

atingir o máximo de 115 milisegundos. O desvio padrão é de 10,32596925, tendo em conta que a média dos resultados é de 66,3.

Pela comparação da análise estatística entre o REST em XML e o REST em Json representada na tabela 8 conclui-se que o REST em Json obtém valores mais baixos quando toca a efetuar pedidos a um endpoint URL.

3.3.3 Testes de Carga aos Web Services

No desenvolvimento de aplicações Web Services torna-se imperativo realizar testes de carga à aplicação, denominados por "LoadTests" pelo SoapUI. Desta forma conseguimos analisar o comportamento e desempenho dos Web Services com diversos pedidos simultâneos a decorrer.

Através do SoapUI também é possível criar testes de carga [SmartBear Software, 2015c] aos Web Services através de um "TestCase". A ideia é instanciar diversas máquinas (utilizadores virtuais) que por sua vez realizam repetidas requisições http ao endereço especificado durante um período de tempo. Para esse período de tempo são especificados a quantidade de acessos simultâneos, o período de tempo, o tempo de atraso, entre outros.

À medida que o "LoadTest" é executado, a tabela de estatísticas "LoadTest" é atualizada continuamente com os dados recolhidos a cada vez que um "TestCase" é executado, permitindo controlar interactivamente o desempenho do serviço (s) alvo, enquanto o "LoadTest" executa.

Testes de Carga ao SOAP e ao REST em XML

No nosso exemplo serão usados 10 *Threads* (utilizadores virtuais) que estarão a aceder aos diferentes *Web Services* durante 160 segundos, com um atraso de 1000 milisegundos. A variação dos resultados depende em grande parte dos BPS (bytes per second) e dos TPS (transactions per second), que pode ser calculado da seguinte forma:

Com base no tempo real passado (padrão):

- TPS: CNT/Segundos que passaram, ou seja, um "TestCase" que foi executado por 10 segundos e foram tratados 100 pedidos receberá um TPS de 10.

- BPS: Bytes/tempo passado, ou seja, um "TestCase" que foi executado por 10 segundos e foram tratadas 100 mil bytes terá uma BPS de 10000.

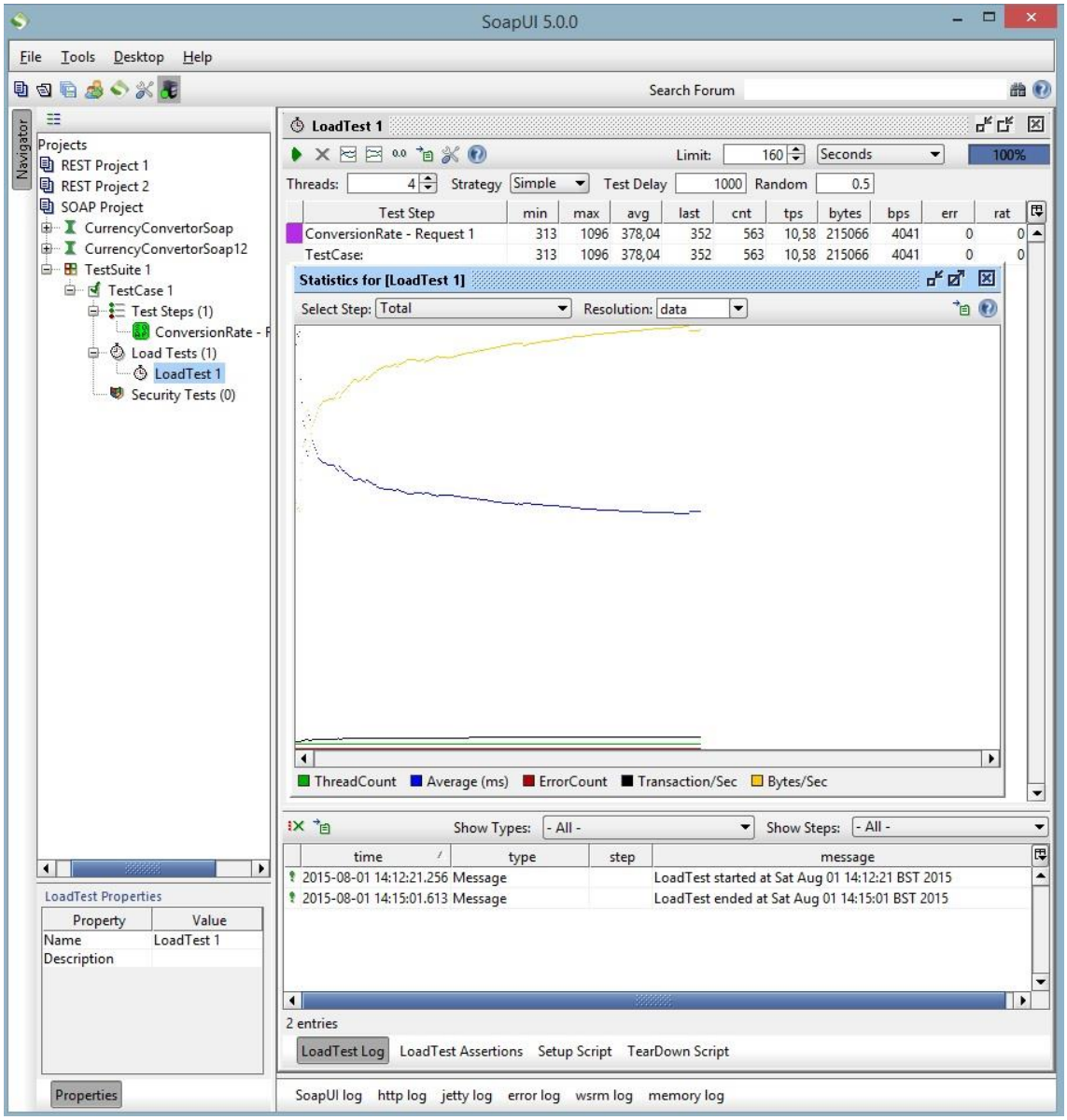


Figura 15 – Gráfico de estatística ao teste de carga efetuado ao SOAP.

No “LoadTest” da figura 15 associado ao “TestCase” do SOAP, a linha amarela (nº de bytes por pedido) tem um crescimento ao longo do tempo bastante grande, como também a linha azul (média) diminui ao longo do tempo. Isto significa que o número de bytes do pedido ao URL tem um aumento significativo consoante o número de threads (utilizadores virtuais) estiverem a aceder, como também o tempo médio de resposta a um pedido aumenta visto que a media do tempo de resposta diminui.

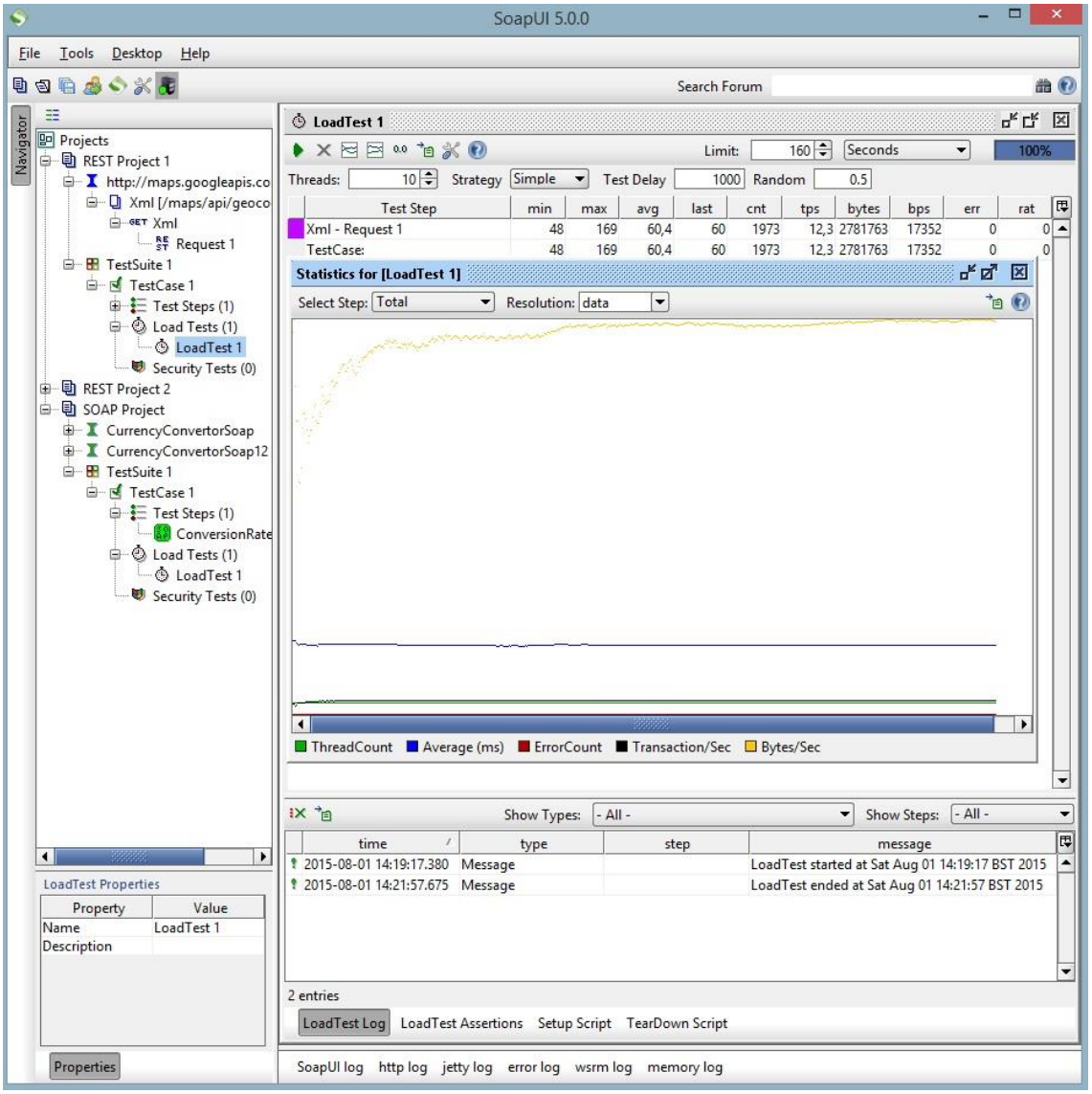


Figura 16 – Gráfico de estatística ao teste de carga efetuado ao REST em XML.

Nesta figura conclui-se relativamente o mesmo em relação à figura 15, só que o tempo médio de resposta a um pedido não diminui tanto como o anterior, daí os resultados obtidos dos pedidos ao feitos ao URL específico são ligeiramente melhores.

Testes de Carga ao REST em Json

No teste de carga para o REST em Json são também usados 10 Threads (utilizadores virtuais) que estarão a aceder aos diferentes Web Services durante 60 segundos, com um atraso de 1000 milisegundos

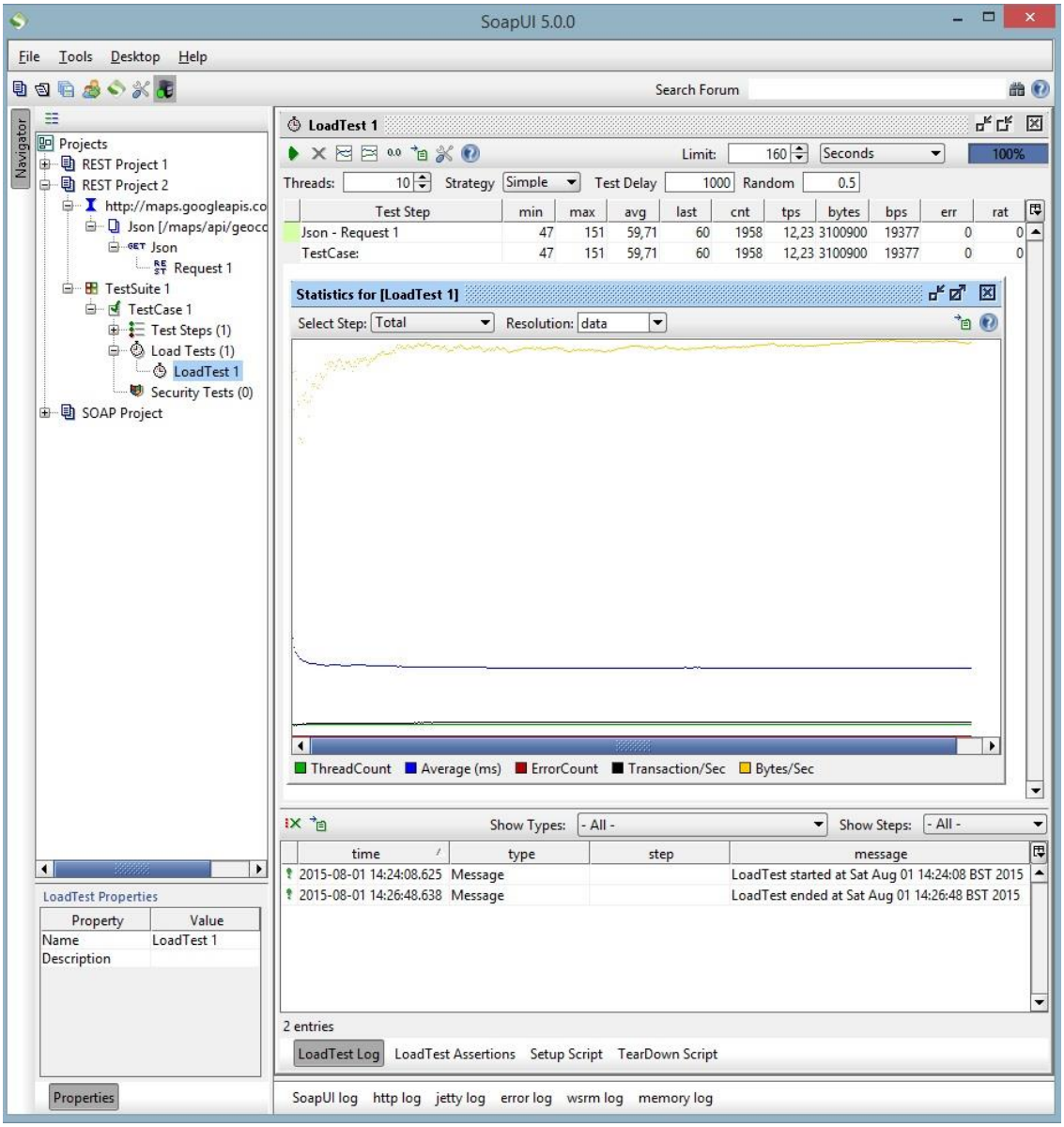


Figura 17 – Gráfico de estatística ao teste de carga efetuado ao REST em Json.

Esta figura é muito semelhante à anterior, só que existe uma ligeira melhoria no tempo médio de resposta ao pedido.

3.4 Conclusões

Através das análises ficou claro que a tecnologia REST fornece um acréscimo de desempenho em relação ao SOAP a nível de tempo de resposta. Já em relação ao formato dos dados REST ou JASON da arquitetura REST houve também algumas melhorias nos tempos de resposta. O REST também é mais simples de usar pelos

programadores, mas se quisermos uma tecnologia mais segura, que apenas justifica-se em casos de operações de banca, neste caso é mais indicado o SOAP.

Segundo o autor John Cowan [John Cowan, 2005], não existem aplicações que se possa pensar que não podem ser feitas para se encaixar nas representações GET/PUT/POST/DELETE. Estas interfaces são suficientemente gerais. Outras interfaces são consideradas prejudiciais porque aumentam os custos de consumo de serviços específicos.

4 Casos de Estudo

A API utilizada na aplicação Beauti tem o seu funcionamento semelhante a outras que também são baseadas na tecnologia REST. Antes do desenvolvimento da API para a Beuti foi decidido analisar outras APIs que se baseiam na mesma tecnologia. Os casos de estudo analisados são o LinkedIn API, o Twitter API, uma vez que se baseiam na tecnologia REST.

É analisado o formato dos dados que as APIs disponibilizam e de que forma uma aplicação pode ir aceder e esses dados.

4.1 LinkedIn API

O LinkedIn [LinkedIn, 2015] é a maior rede social de perfis profissionais. O LinkedIn tem milhões de utilizadores por todo o mundo, e permite ter uma lista de conexões com outros utilizadores. A LinkedIn API permite aos programadores acesso para ler e escrever dados do LinkedIn.

Os pedidod de leitura de dados podem ser feitos utilizando XML ou em JSON, sendo necessário, especificar em que formato queremos retornar os dados. Um exemplo de um pedido GET ao URL: https://api.linkedin.com/v1/people/~, retorna a resposta em XML ilustrada no código 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <id>1R2RtA</id>
  <first-name>Frodo</first-name>
  <last-name>Baggins</last-name>
  <headline>Jewelery Repossession in Middle Earth</headline>
  <site-standard-profile-request>
    <url>https://www.linkedin.com/profile/view?id=…</url>
  </site-standard-profile-request>
</person>
```

Código 13 – Resposta em XML.

É mais conveniente trabalhar com dados em formato JSON, visto ser mais simples de interpretar, para fazer um mesmo pedido GET em JSON apenas basta acrescentar em que formato queremos os dados. Vejamos o seguinte URL: https://api.linkedin.com/v1/people/~?format=json, que retorna a seguinte resposta ilustrada no código 14.

```
{
  "firstName": "Frodo",
  "headline": "Jewelery Repossession in Middle Earth",
  "id": "1R2RtA",
  "lastName": "Baggins",
  "siteStandardProfileRequest": {
  "url": "https://www.linkedin.com/profile/view?id=…"
  }
}
```

Código 14 – Resposta em JSON.

Rest.li

A LinkedIn Api [Joe Betz, 2013] é baseado na framework Rest.li. Rest.li é uma framework *open source* e tem o objetivo de construir arquiteturas de serviço escaláveis e robustas utilizando APIs assíncronas simples.

O LinkedIn precisava em primeiro lugar de uma maneira padrão para descrever os recursos disponíveis na arquitetura orientada a serviços e permitir o acesso de diversos clientes. O objetivo foi padronizar operações de API comuns, enquanto seria permitido que os programadores criassem operações não convencionais. Em segundo lugar era necessário uma solução escalável e assíncrona, porque muitos dos serviços podem receber milhares de consultas por segundo, e foi necessário uma solução para trabalhar sob esse tipo de carga.

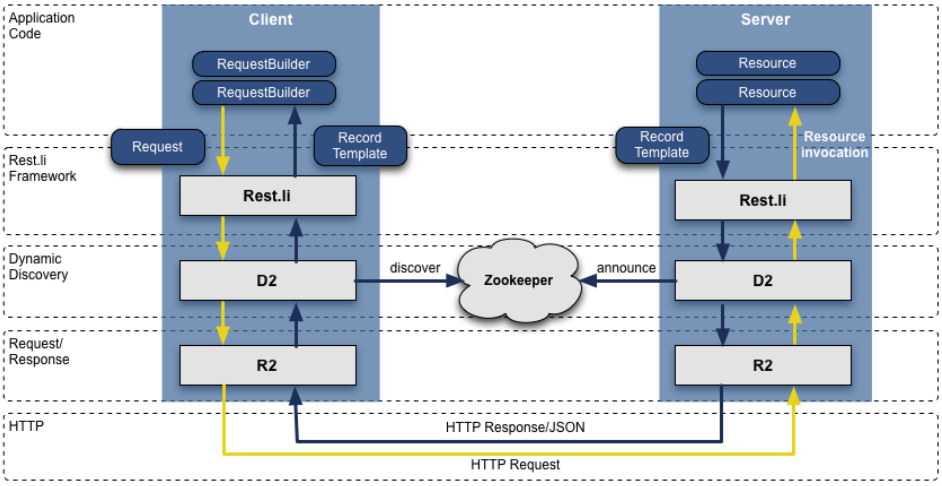


Figura 18 – Rest.li, fluxo servidor-cliente [Joe Betz, 2013].

Rest.li é uma framework Java para construir um serviço estilo REST que usa verbos HTTP tais como o GET e POST para operar as entidades do Rest.li, e um esquema de serialização flexível. Suporta também acesso do tipo seguro a recursos e entidades no cliente e no servidor.

R2 é uma camada de abstração de transporte REST que expõe uma interface de baixo nível estilo REST inspirado por HTTP e uma API assíncrona de alto desempenho, ele também permite mapear operações para transportes não HTTP.

D2 é a camada de descoberta e balanceamento de carga do lado do cliente. É usado o ZooKeeper como um registro para informações sobre os serviços disponíveis e sobre os anfitriões que os fornecem. Os clientes obtêm automaticamente as últimas informações da ZooKeeper e aplicam algoritmos de balanceamento de carga no lado do cliente para distribuir a carga uniformemente entre servidores e reduzir a carga para os servidores.

4.2 Twitter API

O Twitter [twitter, 2015b] é uma rede social de microblogging que permite receber notificações de outros contactos, essas notificações são exibidas no perfil em tempo real. O Twitter API fornece acesso para ler e escrever dados do Twitter, criar um novo Tweet, ler o perfil do autor, os dados dos seguidores, e muito mais.

As respostas no Twitter API estão disponíveis no formato JSON, um exemplo de um pedido GET está ilustrado no seguinte URL: https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2, está ilustrado no anexo C. Este exemplo retorna os mais recentes tweets postados pelo utilizador.

A forma da API aceder aos dados pode ser classificada em dois tipos de arquitetura diferentes:

REST API

Segue uma estratégia pull (pequeno módolo que permite inclusão simples dos dados do twitter) para retornar dados. O login na api faz-se através do OAuth e retorna tokens. É usada para publicar tweets, listar contas, ver o perfil e os seguidores, entre outros. As respostas estão disponíveis em JSON.

Streaming API

A API de streaming fornece um fluxo contínuo de dados do Twitter com baixa latência, o ideal seria enviar tweets sem uma sobrecarga associada. Assim que um pedido de acesso a dados é feito, a Streaming API fornece um stream contínuo de atualizações sem que para isso o utilizador realize um novo pedido.

Para se conectar à Streming API é necessário uma conceção HTTP persistente e em muitos casos não basta usar a API REST. Por exemplo, considerando uma aplicação

web que aceita solicitações do utilizador, e depois faz uma ou mais solicitações à API do Twitter, depois formata e mostra o resultado para o utilizador, como uma resposta à solicitação do utilizador.

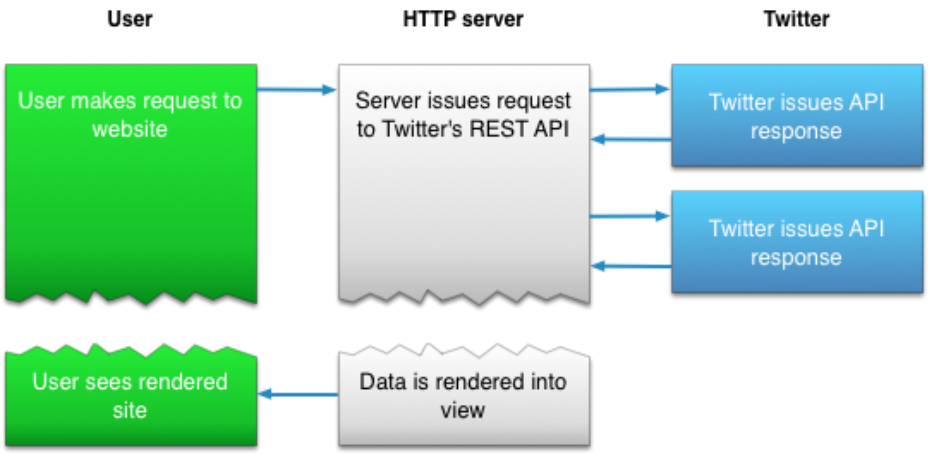


Figura 19 – Twitter REST API [twitter, 2015b].

Tal como ilustrado na figura 19, uma aplicação que se conecta ao Stream API não será capaz de estabelecer uma ligação a um pedido do utilizador. Em vez disso, o código para manter a conexão é executado em um processo separado que lida com solicitações HTTP, tal como na figura 20.

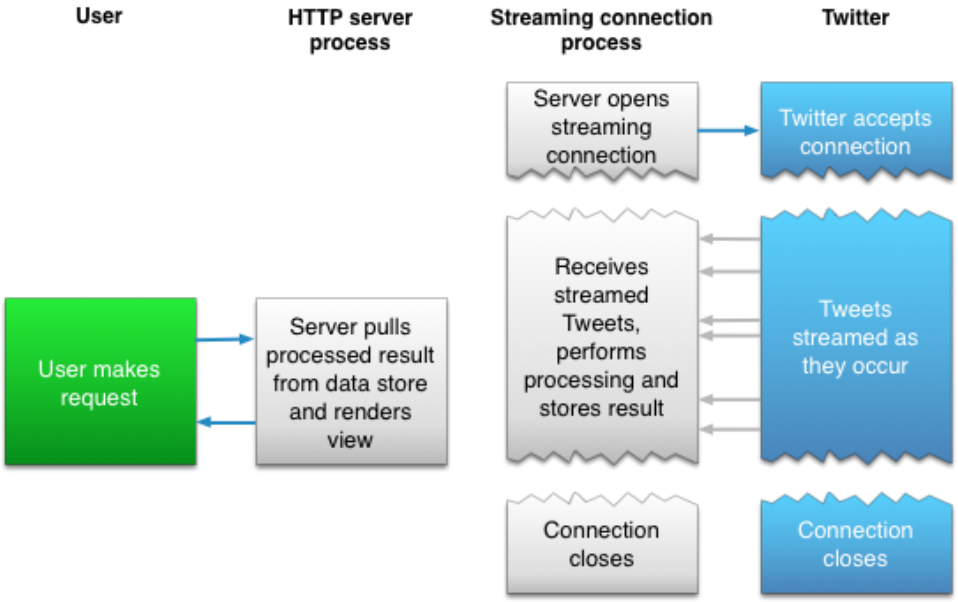


Figura 20 – Stream API [twitter, 2015b].

O processo de streaming recebe os tweets de entrada e executa qualquer análise, filtragem e/ou agregação necessários antes de armazenar o resultado numa

base de dados. O processo de manipulação de HTTP consulta a base de dados para obter resultados em resposta às solicitações do utilizador. Embora este modelo seja mais complexo que o primeiro exemplo, os benefícios de ter um fluxo de Tweet em tempo real torna a integração vantajosa para muitos tipos de aplicações.

4.3 Conclusões

Este capítulo serviu como base para o estudo de aplicações com o mesmo género de funcionalidades e funcionamento da aplicação Beauti.

Através deste capítulo foi possível retirar algumas conclusões tais como, quase todas as API analisadas são semelhantes no seu funcionamento, isto é, fazem autenticação através de pedidos a um Web Servidor utilizando a arquitetura REST e utilizam o formato de mensagens JASON, a não ser que seja necessário haver um maior rigor a nível de segurança, nesse caso pode ser usado SOAP que é mais adequado.

O funcionamento da aplicação Beauti baseou-se nestes princípios, autenticar-se a um servidor para tecnologia REST, pedir uma autenticação ao utilizador e usar o formato de mensagens em JASON.

5 Análise e conceção da aplicação e da API baseada em REST

Este capítulo é dedicado ao projeto que serviu como pilar desta tese de mestrado, e que motivou o uso da tecnologia REST para acesso a um Web Service. O nome da aplicação criada é "Beauti", uma vez que esta tem como principal finalidade a marcação de serviços de beleza e bem-estar por parte do utilizador.

Neste capítulo, em primeiro lugar são analisados os requisitos da aplicação detalhados no capitulo 1.3. Depois é analisada a arquitetura do sistema, em terceiro é detalhada a API usada (BAPI) para comunicar com o servidor. Depois são analisados os formatos de um pedido GET, e a segurança baseada em tokens. Depois são analisadas as diferentes interfaces através de layouts. E finalmente são analisados alguns dos diagramas de estados, nomeadamente os mais relevantes para a aplicação.

São também feitos testes de compatibilidade e usabilidade que têm como propósito testar a aplicação em vários dispositivos móveis. Aqui foram testadas a fluidez da aplicação em relação à interface.

5.1 Requisitos do sistema e do utilizador

Aqui serão analisados os requisitos detalhados da aplicação com base nos objetivos apresentados no capítulo 1.3.

A aplicação tem 2 páginas importantes: página de pesquisa (homepage) e página de marcações, e outras páginas secundárias, sendo elas: página de login, página de registo, e página de configurações. As restantes opções do menu lateral esquerdo são meros links para URLs importantes para a aplicação.

De seguida são detalhados os requisitos de cada página, ou seja, o conteúdo da página e funcionalidades que deve conter.

Página de login:

Esta página deve conter o logotipo da Beauti, um botão para fazer login através do Facebook, e um formulário de login (email e password). Este formulário impede o preenchimento de dados inválidos, tais como campos vazios, ou dados que não existam no servidor.

Página de pesquisa (homepage):

Nesta página deve ser mostrada uma barra de pesquisa que exibe uma lista de estabelecimentos profissionais após feita a consulta pelo utilizador. Cada elemento listado deve conter: 1) Uma imagem do profissional ou do estabelecimento; 2) O nome do estabelecimento; 3) O local do estabelecimento.

Página para efetuar marcação:

Esta página é uma página que é visualizada pelo utilizador após este fazer a pesquisa por um nome do profissional ou pelo estabelecimento. A efetuação da marcação tem como finalidade efetuar a marcação por parte do cliente num profissional de saúde/beleza à sua escolha, escolher o tipo de serviço que deseja, após o que pode escolher uma data juntamente com a hora, e agendar.

Página para ver marcações:

Esta página deve conter uma lista de marcações que foram feitas pelo utilizador. Cada elemento da lista deve ser constituído pelos seguintes campos: 1) Nome do pofissional no canto esquerdo; 2) Nome do salão no canto esquerdo; 3) Data e hora em que a marcação foi feita no canto direito; 5) Conteúdo da marcação, indicando o tipo de serviço que será realizado; 6) Diferentes botões, dependendo se o profissional confirmou ou não a marcação.

Página de configurações:

Nesta página deve ser possível ver e alterar todos os detalhes que dizem respeito ao profissional, tais como: 1) Foto; 2) Nome; e 5) Telefone. O endereço eletrónico também é listado, mas não pode ser alterado.

5.2 Contextualização

O projeto desenvolvido no âmbito desta tese está inserido num contexto empresarial, visto ter sido proposto pela empresa Beauti.

O funcionamento da ideia está ilustrado na figura 1 e baseia-se em: 1) encontrar profissionais de saúde/beleza; 2) escolher o tipo de serviço desejado (e.g. corte de cabelo, manicura, pédicure, etc.); 3) escolher um dia e a hora da marcação; 4) receber notificações das marcações efetuadas para o correio eletrónico.

Figura 21 – Funcionamento da ideia

Este tipo de serviço pode ser usado pelos profissionais de saúde/beleza (p.e. recebem reservas), como também por utilizadores (p.e. efetuar reservas), e adapta-se a diferentes tipos de negócio, tal como cabeleireiros, centros de estética, centros de depilação a lazer, institutos de beleza, spas, especialistas em massagens, especialistas em yoga, médicos e psicólogos, dermatologistas, nutricionistas, centros de depilação a lazer, cosmetologistas, treinadores de fitness, fisioterapeutas entre outros.

A vantagem deste género de serviço ilustrada na figura 2 é permitir marcações a qualquer hora e de qualquer lugar, lembretes automáticos relacionados com a marcação, emissão de copões de desconto e mensagens automáticas de aniversário.

Figura 22 – Vantagens do serviço

O uso de uma aplicação móvel para realizar as marcações traz vantagens que vão de acordo com a política da empresa, no sentido em que torna possível realizar as mesmas em qualquer hora e qualquer lugar.

A criação de uma aplicação para efetuar as reservas num profissional de saúde/beleza prende-se com o forte crescimento na área de dispositivos móveis e o uso de aplicações para a realização de múltiplas tarefas, tais como fazer reserva de um voo, consultar o meu correio eletrónico, reservar um hotel, entre outras.

5.3 Arquitectura do sistema

Neste capítulo é abordada a arquitetura do sistema, representada na figura 31. Esta arquitetura usa o modelo MVC (Modelo, Vista e Controle) com o objetivo de dividir

as funcionalidades do sistema em camadas, e reduzir um problema maior em problemas mais pequenos.

Segundo Daniel Bastos [Daniel Flores Bastos, 2011] a utilização do MVC prende-se com o facto de querer reduzir a complexidade do sistema, este tipo de arquitetura tem como objetivo dividir um problema maior em problemas mais pequenos e de menor complexidade. Dessa forma, qualquer tipo de alterações em cada uma das camadas não interfere nas outras, o que facilita a atualização de *layouts*, alteração nas regras e adição de novos recursos. Em grandes projetos, o MVC facilita muito a divisão de tarefas entre a equipa: 1) Facilita o reaproveitamento de código; 2) Facilita na manutenção e adição de recursos; 3) Permite uma maior integração da equipe e/ou divisão de tarefas; 4) Já existem diversas tecnologias que têm adotado esta arquitetura; 5) há uma facilidade em manter o código sempre limpo.

Na arquitetura da figura 23, o MVC está organizado em Vista utilizado na aplicação, e em Modelo e Controlo utilizados na BAPI. Segundo esta arquitetura, a aplicação faz um pedido HTTP ao BAPI, o BAPI é responsável pelo controlador e pelo modelo: 1) O controlador carrega um modelo e efetua um método que realiza a validação; 2) No modelo são efetuados as tarefas: -Armazena as informações digitadas pelo utilizador; -Realiza a consulta. Retorna verdadeiro em caso de sucesso, ou falso em caso de inválido; 3) O controlador verifica o que o modelo retornou: -Se retornar verdadeiro guarda as informações e retorna para a Vista; -Se retornar falso redireciona o utilizador para a tela inicial retornando a mensagem

A arquitetura do sistema da figura 23 seria a arquitetura ideal e iria funcionar da seguinte forma: 1) A aplicação pode fazer um pedido a um servidor de imagens (fornecedor externo) através de um URL; 2) A aplicação pode fazer um pedido HTTP à Beauti API, esta por sua vez pode abrir um canal de comunicação com a base de dados da Beauti através de Sockets/TCP. Os fornecedores externos (e.g. API's externas) podem fazer pedidos HTTP à Beauti API. Actualmente ainda não existe um servidor externo de imagens, ou seja, as imagens estão alojadas na base de dados da beauti, em relação ao resto da solução mantêm-se.

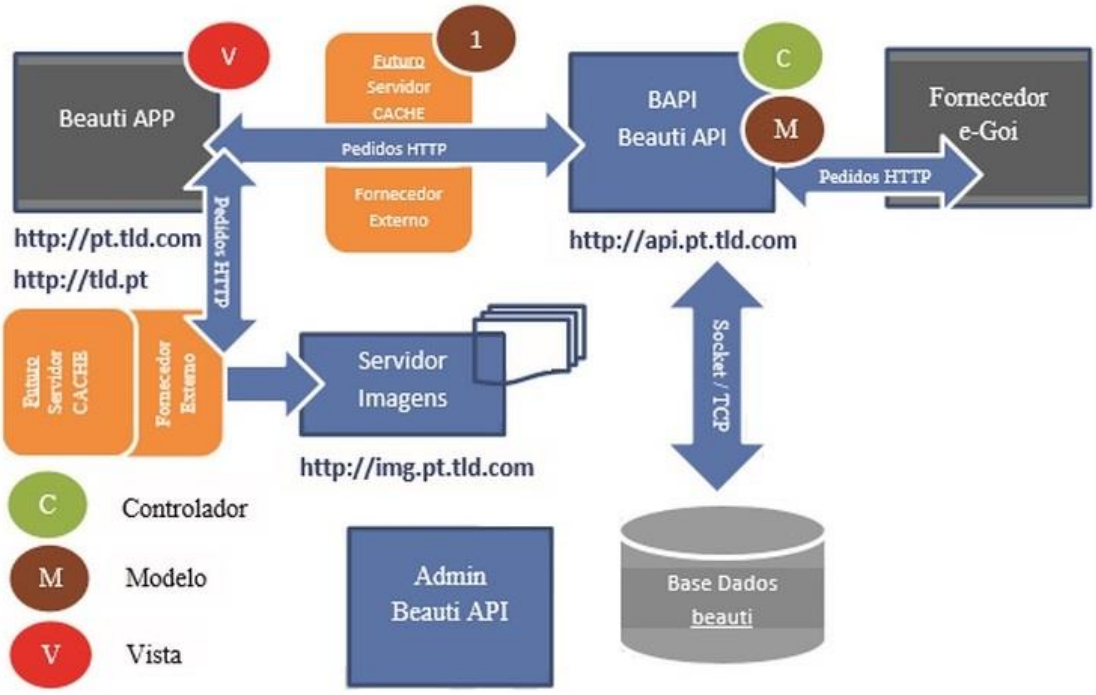


Figura 23 – Arquitetura ideal do sistema.

A Beauti APP pode ser considerada uma view entre os dados da base de dados e imagens. A BAPI é invocada através de endpoints compostos por uma estrutura genérica, exemplo api.beauti.pt/versao/model/controlador.formato, desta forma podem existir vários métodos diferentes que controlam o que queremos obter como resultado (view). Existe uma autenticação para determinados endpoints através de um token enviado como parâmetro para chamadas GET ou POST. O token é obtido após o login. A api da beauti também pode gerar resultados em alguns formatos como XML ou JSON, essa especificação é posta no final do método, por exemplo api.beauti.pt/1/users/getusersdetails.json?token=xyz. Os resultados da api são por norma pares de atributos de determinados objectos relevantes para esse endpoint, podendo ser textos ou números ou *links* para imagens. Os links das imagens são gerados pela api, por exemplo a foto de utilizador é gerada como um *link*, que basicamente é um endereço de identificação de imagem binária que existe armazenada na base de dados como blob. Poderia ser utilizado um servidor externo em cloud que armazena todas as imagens de forma semelhante, mas essa arquitetura ainda não foi possível. É importante distribuir os recursos, imagens, css, JavaScripts por vários domínios para aumentar a performance dos pedidos, limitados por um *browser*. Basicamente o diagrama mostra que o ponto central é a api que comunica com a base de dados de informação dos utilizadores e também com serviços externos como por exemplo o egoi, para enviar sms via texto atraves da bapi. Mas a api também pode fornecer urls para

outro banco de dados externo de imagens. A app é meramente uma view ou interface de interacção com a api. Que nos fornece todas as informações necessárias.

5.4 Descrição da Beauti API – REST (BAPI)

A opção por uma API REST permite tornar o sistema mais robusto, escalável e menos dependente de eventuais alterações que ocorram na camada de apresentação.

A arquitetura adotada nesta solução compreende a total separação das 3 camadas da plataforma: Dados, Lógica aplicacional e a camada de apresentação. Desta forma e utilizando a API disponibilizada poderemos ter a plataforma Beauti, em diversos dispositivos. A API REST da Beauti suporta a camada de dados e lógica aplicacional da aplicação e é o cerne do sistema.

5.4.1 Nomenclaturas e normas gerais

Este subcapítulo diz respeito nomeadamente às regras ou metodologias usadas pelo Beauti API.

Pedidos GET : será utilizado de uma forma geral para operações de listagem de dados. Como por exemplo listar utilizadores.

Pedidos POST: será utilizado de uma forma geral para operações de inserção e atualização de dados.

Todos os pedidos HTTP possuem ainda um parâmetro com a versão da API a ser pedida. Desta forma e de futuro a API poderá ser alterada sem prejuízo de eventuais clientes que já possuam implementações em função de versões anteriores.

Todos os pedidos HTTP possuem um parâmetro de formato de resposta, que permite especificar o tipo de resposta esperada pelo serviço. Para já disponibiliza-se formato, JSON ou XML. Outros formatos poderão vir a ser disponibilizados em função de pedidos.

O URL base para a api da plataforma beauti sera então api.pt.tld.com

A língua dos pedidos da API segue duas normas ISO (CODIGO_PAIS-CODIGO_IDIOMA). Para o código do pais, a norma é a ISO 3166, para o idioma a norma é a ISO 639-1. Exemplo: pt-PT. O nome do parâmetro é “language”. Sempre que o

parâmetro língua não seja fornecido, será assumido por defeito que a língua é "Português".

5.4.2 Descrição dos métodos da API

Nesta secção ficarão documentados os métodos da API que estão disponíveis para utilização:

a) Todos os métodos e respetivos parâmetros deverão ser chamados/utilizados em lowercase;

b) Todos os pedidos a API poderão levar um parâmetro que deverá ser incluído no header do pedido, designado por APPKEY. Esta appkey irá permitir contabilizar o número de acessos feito por cada uma das aplicações à plataforma BAPI. Para além disso este parâmetro serve para controlar o acesso aos métodos da API;

c) Sempre que um método retorna um número significativo de resultados é utilizado um mecanismo de paginação. Em qualquer método é possível passar a seguinte informação: -Perpage: número de registos a devolver em cada página. Por defeito a API assume 10; -Page: O número da página a obter por parte do método da API;

d) Todos os métodos aquando de um erro ou de um comportamento inesperado, resultante de um input em falta, ou outa situação, retornam um item chamado "error" que contem o código do erro "ID" e a sua respetiva descrição "DESCRIPTION" na língua do pedido efetuado;

e) Estes métodos permitem ter toda a funcionalidade para agentes ou clientes baseados num modelo que garante que podemos depois evoluir possíveis aplicações cliente, como por exemplo as móveis.

Esta secção está organizada da seguinte forma: a) Verbo (GET ou POST) Nome objeto / Nome Função (ou método); b) Descrição Sucinta método; c) URL onde este método está disponível; d) Indicação se necessita de autenticação; e) Formatos de resposta disponíveis; f) Métodos HTTP disponíveis; g) Indicação de Versão atual; h) Parâmetros; i) URL/Link Exemplo e Resposta Exemplo.

5.5 Formato de dados dos pedido GET

Esta secção serve para perceber como é feita a solicitação e em que formato veem os dados da API, como também para ilustrar exemplos dos pedidos.

A solicitação de dados da API dá-se através de uma tarefa assíncrona (AsyncTask), e irá retornar informações no formato de JSON.

Login

O login é feito por um pedido GET a um URL tem o seguinte formato http://api.beauti.pt/1/users/login.json?username=**username**&password=**password**&lang=pt-pt, e a descrição das variáveis está ilustrada na tabela seguinte.

Tabela 7 – Descrição das variáveis do login.

Variáveis Query Url	**Descrição**
Username	Nome de utilizador, geralmente endereço de correio eletrónico.
Palavra-chave	Senha para a consulta login.

Um exemplo de um pedido GET ao URL anterior retorna a resposta ilustrada no código 17.

```
{
  "login": "1",
  "usertype": "agent",
  "acess_token": "7739e75170aed9382a1ac58f4df6c4d8553e9ca4adca88.78311822",
  "superuser": "0",
  "error": {
    "id": "0",
    "description": ""
  },
  "memory": "1289496"
}
```

Código 17 – Login: Resposta ao pedido.

A tabela 8 ilustra os valores possíveis que o pedido pode retornar.

Tabela 8 – Valores possíveis da resposta.

Valor	**Sucesso**	**Falha**
Login	1	“”, Falha no login em caso de credenciais erradas.
Token	Retorno do token se o login estiver ok	-

Registar – registerUser

O registo é feito por um pedido POST ao seguinte URL http://api.beauti.pt//1/Users/registerUser.json, os valores preenchidos são enviados através do nameValuePair que é um par de valores <Key, Value>, nomeadamente o campo a alterar e o novo valor. Na tabela seguinte são apresentados os valores das variáveis para onde serão enviados os valores que serão alterados

Tabela 9 – RegisterUser: Descrição das variáveis enviadas pelo POST.

Variáveis POST	**Descrição**
firstname	Primeiro nome de utilizador.
lastname	Último nome de utilizador.
username	Email do utilizador.
password	Password do utilizador.

Configurações – GetUserDetails:

A obtenção dos dados do utilizador é feito por um pedido GET ao seguinte URL http://api.beauti.pt/1/users/getuserdetails?token=**token**, e a descrição das variáveis está ilustrada na tabela seguinte.

Tabela 10 – Descrição das variáveis do getUserDetails.

Variáveis Query Url	**Descrição**
Token	Token de acesso, obtida no login e única para cada utilizador.

Configurações – editUser:

Para alterar os dados do utilizador é feito um pedido POST ao seguinte URL http://api.beauti.pt/1/Users/editUser.json, em que são enviados os valores que queremos alterar através do nameValuePair. Na tabela seguinte são apresentados os valores das variáveis para onde serão enviados os valores que serão alterados.

Tabela 11 – EditUser: Descrição das variáveis enviadas pelo POST.

Variáveis POST	Descrição
id	Identificação do utilizador.
token	Token de acesso.
fname1	Primeiro nome de utilizador.
fname2	Último nome de utilizador.
fcodp1	Primeiros 4 dígitos do código postal.
fcodp2	Últimos 3 dígitos do código postal..
telemovel2	Telemóvel do utilizador.

Um exemplo do pedido POST ao URL anterior retorna a resposta ilustrada no código 18.

```
{
 "editUser":"1",
 "error":{"id":"1062","description":"Duplicate entry '3024-3024' for key
 'unique'"},
 "speed":"0.014071941375732","memory":"1289144"
}
```

Código 18 – editUser: Resposta ao pedido.

Ver Marcações – getMarcaActual:

A visualização das marcações é feita por um pedido GET ao seguinte URL http://api.beauti.pt/1/marcacoes/getMarcaActual.json?token=**token**, aqui o token tem a mesma descrição que na tabela 10.

Pesquisa

A pesquisa é feita através de um pedido GET ao URL http://api.beauti.pt/1/agentes/list?nome_agente=**agente**&orderby=4&perpage=5&page=**page**

Tabela 12 – Valores possíveis da resposta.

Variáveis Query Url	Descrição
Agente	Nome de agente.
Orderby	Ordem dos resultados.
Perpage	Número de resultados por página, no meu caso escolhi 5 por definição.
Page	Número da página em que está.

Fazer Marcação – doMarcacao

Para fazer uma marcação é feito um pedido POST ao seguinte URL http://api.beauti.pt//1/marcacoes/doMarcacao.json, os valores preenchidos são enviados através do nameValuePair. A tabela seguinte ilustra as variáveis necessárias para que o pedido se efetue com sucesso.

Tabela 13 – DoMarcacao: Descrição das variáveis enviadas pelo POST.

Variáveis POST	Descrição
token	Token de acesso.
date	Data em que é feita a marcação.
contacto	Contacto da pessoa que faz a marcação.
idprofessional	Identificação do profissional onde é feita a marcação.
total_servicos	Número de serviços a realizar.
servico_0	Identificação do serviço a realizar.

5.6 Segurança na aplicação baseada em tokens

Tal como foi abordado no capítulo 2.11, a segurança é um aspeto fundamental numa aplicação que envolva dados pessoais de um utilizador, porque se a segurança não existir torna-se mais fácil a um hacker aceder a esses mesmos dados.

Embora o hashing da password seja um bom método para proteger o utilizador final, é um método com alguns problemas. Foi necessário então encontrar uma alternativa igualmente segura e que não causasse problemas, tal como a utilização de tokens.

Em java, a utilização de tokens dá-se da seguinte forma: 1) Fazemos login via API e obtemos o token, este token é armazenado numa variável qualquer para ser posteriormente ser usado nas próximas chamadas; 2) O token é o que identifica o utilizador; 3) O token tem um tempo de expiração/validade, que depende; 4) Podemos interromper o acesso (webmaster) a qualquer momento desativando aquele específico token.

5.7 Modelação e criação da interface

A interface criada foi desenvolvida e testada para Android. Apesar de ser uma aplicação com funcionalidades simples, um dos objetivos era utilizar a tecnologia REST para obter informações de um servidor disponibilizado na internet (Web Service).

A modelação da interface da aplicação foi feita de acordo com muckups de alta-fidelidade disponibilizados pela Beauti, que não são mais do que esboços do que pretendemos que a interface da aplicação seja que depois foram transformados em layouts finais. Felizmente não foi necessário validar a interface junto do cliente, visto que foi o mesmo que a criou e disponibilizou. O cliente teve em conta, embora na conceção da interface, a aparência, simplicidade e usabilidade para o utilizador final da aplicação.

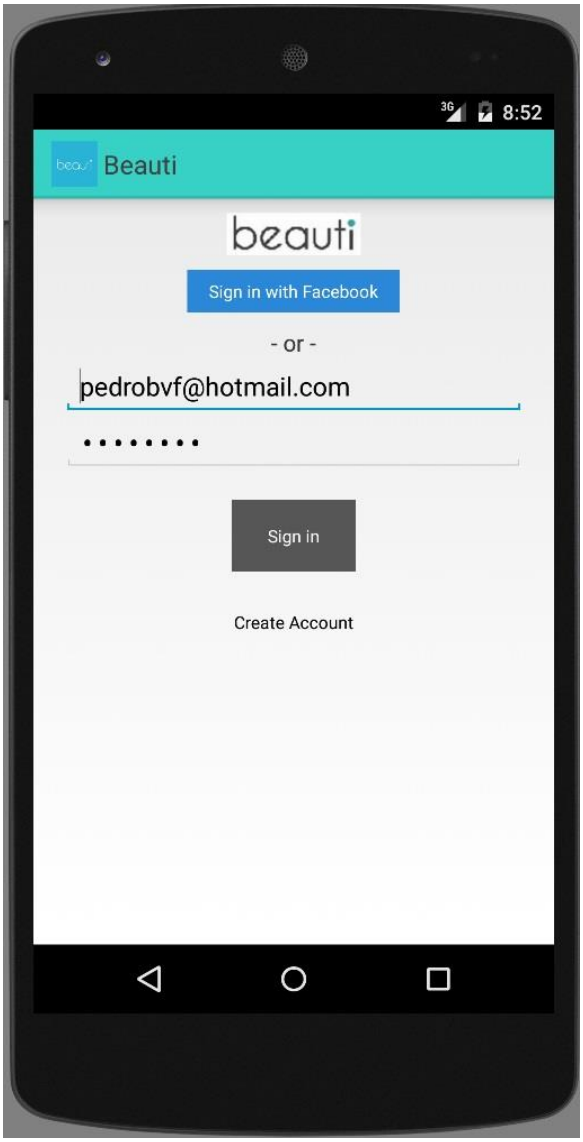


Figura 24 – Layout: Login.

A figura 24 corresponde à página de login. Esta página permite ao utilizador autenticar-se através das contas do Facebook, ou autenticação através do preenchimento de um formulário (email e password). Desta página podemos navegar para outras, como o registo, ou a página inicial da aplicação. Para o desenho desta página foram usados os seguintes componentes gráficos do Android: ImageView, TextView, EditText, Button, e ProgressBar.

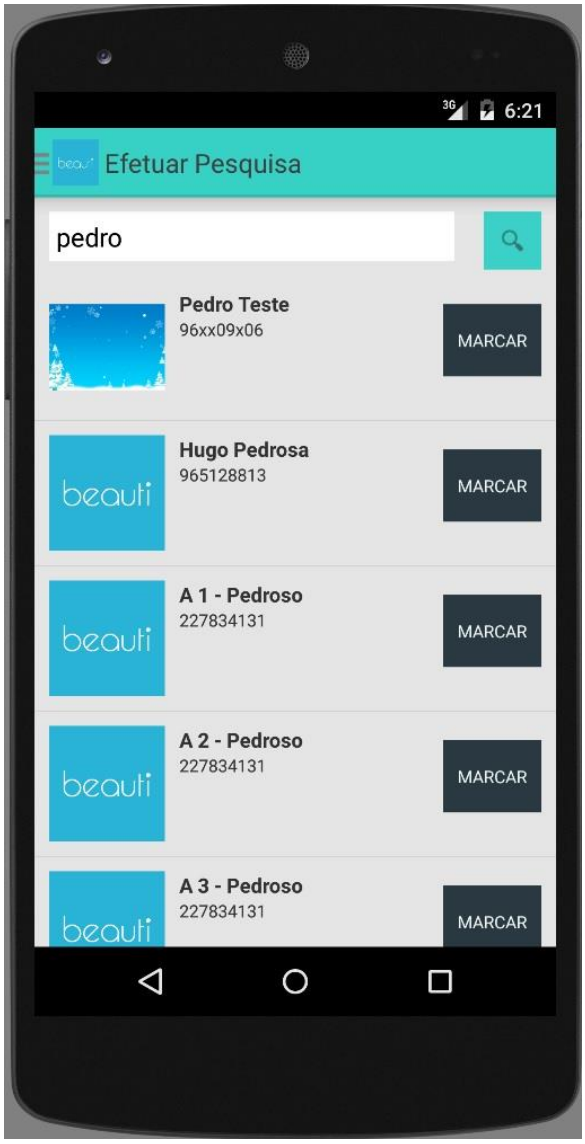


Figura 25 – Layout: Pesquisa (home).

Na figura 25 podemos ver a página inicial da aplicação. Nesta página é apresentado ao utilizador duas barras de pesquisa para que este possa pesquisar o estabelecimento, profissional ou serviço que deseja. Após feita a pesquisa e retornar os resultados, aparece uma lista de profissionais que ao clicar em cima redireciona o utilizador para a criação de uma marcação. Para o desenho desta página foram usados os seguintes componentes gráficos do Android: EditText, ImageButton, ProgressBar, ListView, ImageView, TextView, Button.

Figura 26 – Layout: Efetuação de marcação.

A figura 26 diz respeito à efetuação da marcação, onde aparece uma imagem correspondente ao profissional que o utilizador pesquizou, juntamente com os serviços disponibilizados pelo mesmo e um botão de reserva. Para o desenho desta página foram usados os seguintes componentes gráficos do Android: EditText, ImageButton, ProgressBar, ListView, ImageView, TextView, e Button.

Figura 27 – Layout: Ver Marcações.

Na figura 27 acima representada podemos ver o layout para ver as marcações, sendo logo percetível que nesta página será exibida uma lista de marcações efetuadas. Dando a possibilidade ao utilizador de cancelar uma das marcações que tenha efetuado. Para o desenho desta página foram usados os seguintes componentes gráficos do Android: ProgressBar, ListView, TextView, Button.

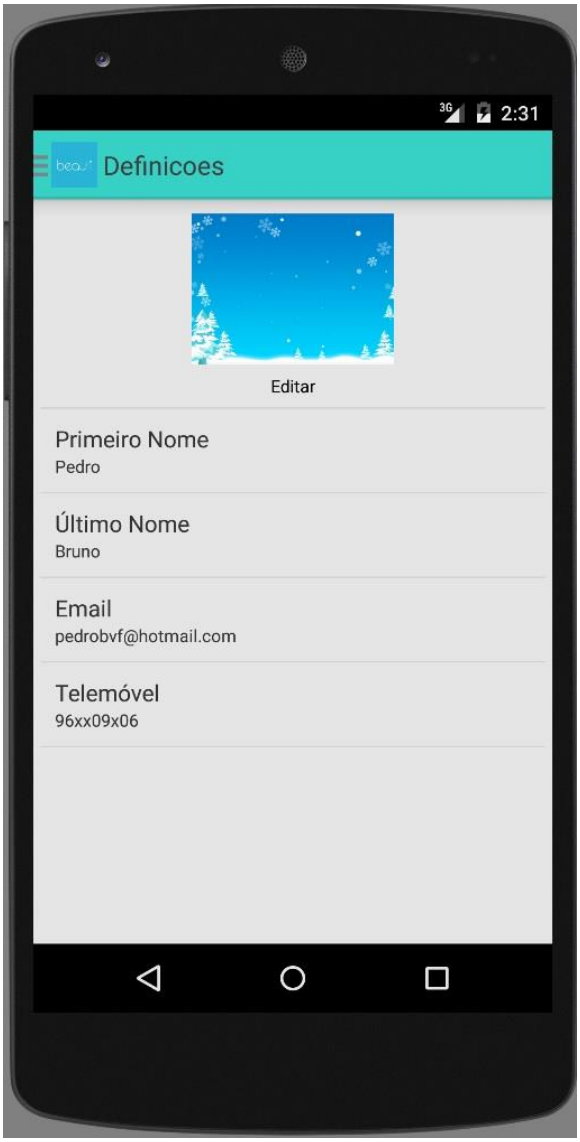


Figura 28 – Layout: Definições.

A figura 28 corresponde à página de configurações. O objetivo desta página é ver e alterar todos os detalhes que dizem respeito ao utilizador: como a foto, os nomes, e o nº telemóvel. Embora seja possível ver o *email*, não é possível alterar-lo. Para o desenho desta página foram usados os seguintes componentes gráficos do Android: ProgressBar, ImageView, TextView.

5.8 Diagramas de estados da interface

Neste capítulo são apresentados alguns diagramas de estados correspondentes a interfaces da aplicação, nomeadamente os diagramas correspondentes ao login e à marcação em um profissional de saúde/beleza.

Nestes diagramas são especificados alguns estados possíveis das interfaces, bem como as transições entre as páginas. A escolha de apenas estes dois diagramas prende-se com o facto de serem ações importantes na aplicação.

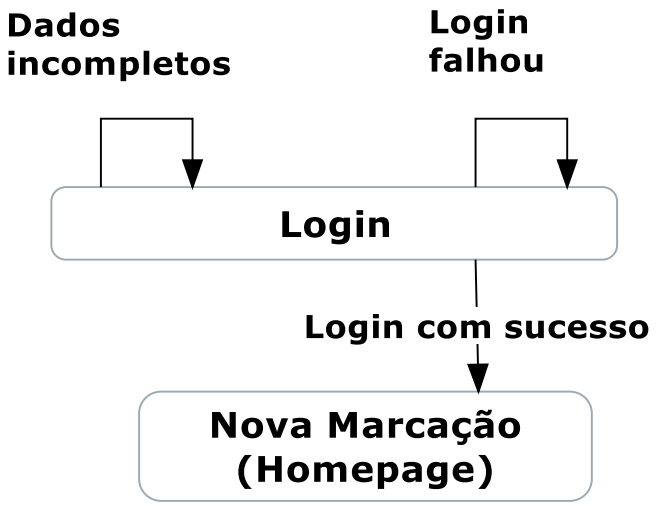


Figura 29 – Diagrama de estados do login

A figura 29 corresponde á página de login, onde inicialmente o utilizador é direcionado para a página do login ou para a página *homepage* conforme já estiver conectado.

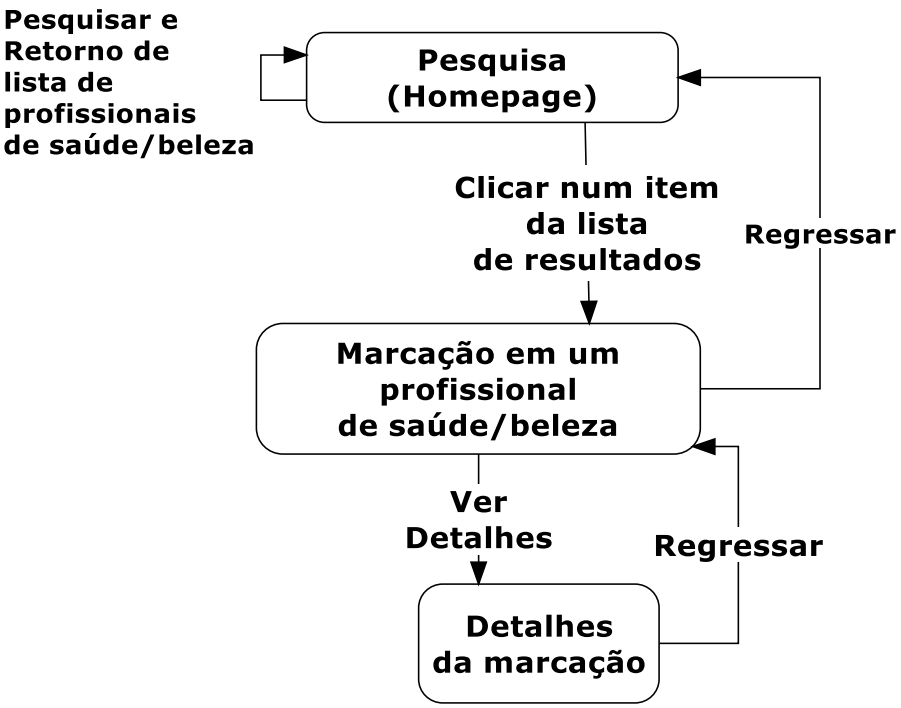


Figura 30 – Diagrama de estados da marcação em um profissional.

A figura anterior corresponde à *homepage*, nesta página é dada a possibilidade de o utilizador pesquizar por um estabelecimento ou código postal retornando, após isso, a uma lista de profissionais com determinado nome.

Após clicar em um desses profissionais, o utilizador é direcionado para a página de marcação onde aparece os tipos de serviços que determinado profissional oferece. Após o utilizador clicar num botão e confirmar a marcação, é possível verificar os detalhes da mesma.

5.9 Diagrama de sequencia

O diagrama de sequência tem com o objetivo representar a sequência de passos, ou de mensagens passadas entre o utilizador, as páginas da aplicação e o servidor.

A figura seguinte ilustra alguns dos passos efetuados entre o utilizador e as partes mais importantes da aplicação.

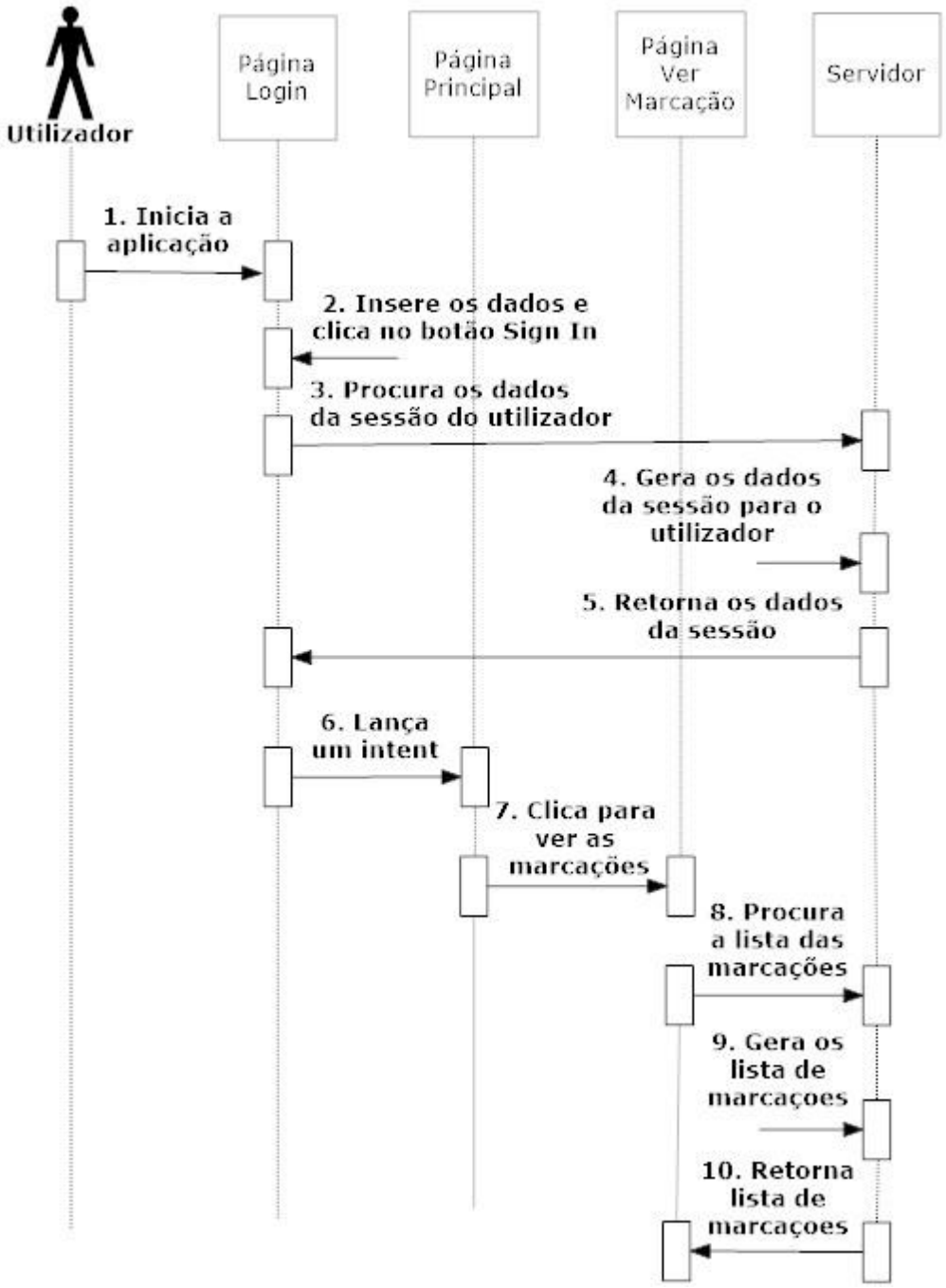


Figura 31 – Diagrama de sequência.

5.10 Testes de compatibilidade e usabilidade

Estes testes têm como finalidade testar a aplicação em vários dispositivos móveis com diferentes versões androide, com o propósito de assegurar o seu correto funcionamento e visualização.

A aplicação foi testada em vários ecrãs de vários dispositivos, aqui os emuladores virtuais fornecidos pelo eclipse tiveram um papel importante, pois permitiram criar dispositivos com as características desejadas, nomeadamente com ecrãs de tamanhos diferentes. Os emuladores ou dispositivos utilizados para realizar os testes foram: 1) WXGA (Tablet) – versão 4.4.2; 2) Nexus 5 – versão 4.1.2; 3) Nexus S – versão 3.2; 4) Nexus One – versão 2.1.

Em relação aos testes realizados em emuladores, esses tiveram como objetivo verificar a correta visualização da aplicação em resoluções de ecrãs diferentes, os ecrãs foram: 1) 1280x800; 2) 1080x1920; 3) 720x1280; 4) 480x800. Alguns dos resultados foram positivos, isto porque embora a aplicação se adaptou a essas resoluções, nas versões android mais antigas as funcionalidades não são suportadas. Os parâmetros tidos em conta para estes testes foram: a) correto *layout* das páginas; b) layout adapta-se; c) suporte do menu lateral esquerdo.

Tabela 14 – Testes de Compatibilidade

	WXGA (Tablet) (1280x800)	Nexus 5 (1080x1920)	Nexus S (720x1280)	Nexus One (480x800)
Correta visualização das páginas	Sim	Sim	Não	Não
Layout adapta-se	Sim	Sim	Sim	Sim
Suporte do menu lateral esquerdo	Sim	Sim	Não	Não

5.11 Conclusões

Através deste capítulo é possível concluir que foi atingido o cumprimento dos requisitos especificados no capítulo 1.2, como também é possível verificar que a interface mostrada já tinha sido validada pelo cliente quando foi iniciada.

Em relação ao sistema de segurança e à arquitetura, foi tido algum cuidado por parte do cliente na conceção das mesmas de forma a garantir segurança e facilidade no acesso aos dados por parte da aplicação.

6 Modelo de negócio

O modelo de negócio desempenha um papel importante para o sucesso de uma empresa ou produto, porque explica a forma como gerar valor.

Este capítulo tem como principal objetivo perceber como o produto pode criar valor para o público, ou seja, permite otimizar o processo de marcação, aumentar o tempo livre das pessoas porque não têm de esperar no local pela marcação e permite uma melhor gestão do horário.

Neste capítulo é analisado o público-alvo que a aplicação pretende atingir, o produto e respetivas vantagens e desvantagens, e finalmente como será feita a venda do produto.

6.1 Conceito do modelo de negócio

Segundo Sandro Santos [Sandro Santos, 2014], o modelo de negócio é uma ferramenta para estruturar negócios, com o objetivo de descrever o que é relevante para um negócio.

O modelo de negócio pretende responder a algumas questões tais como:

- Como pretendemos fazer dinheiro?
- Quais os gastos que teremos para fazer o produto?
- Quem são os clientes?
- De que forma vai chegar o produto aos clientes?

A figura 30 ilustra as 9 áreas em que Osterwalder [Alexander Osterwalder, 2009] dividiu o negócio.

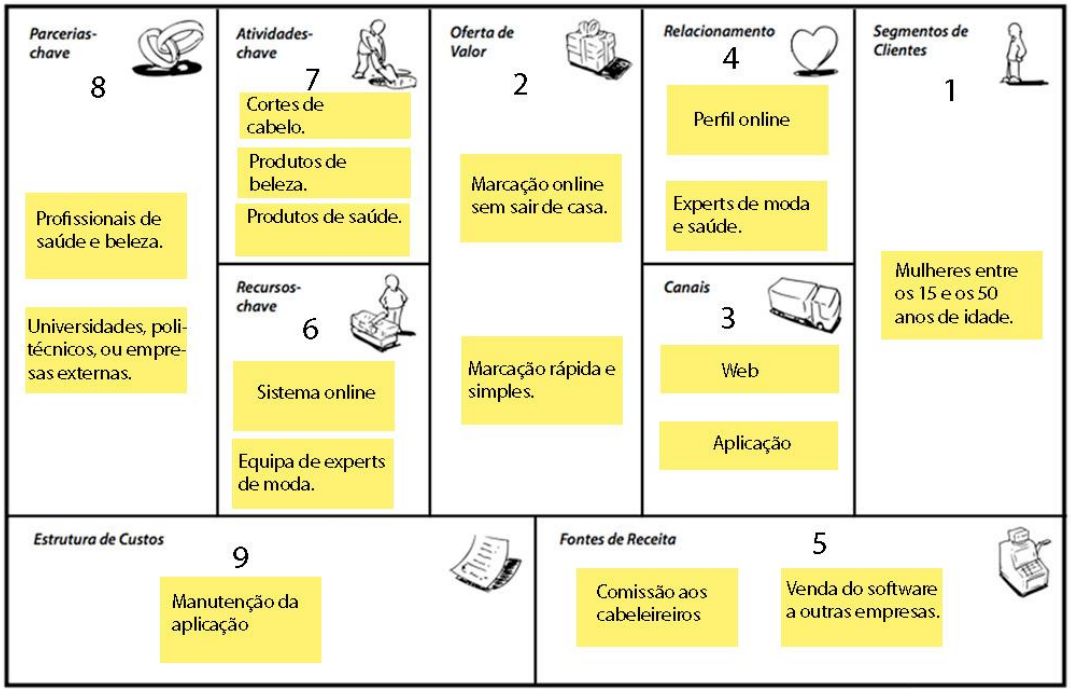


Figura 32 – Quadro do modelo de negócio [Sandro Santos, 2014].

A tabela seguinte permite perceber cada elemento do quadro de negócio.

Tabela 15 – Descrição dos blocos do modelo de negócios.

Elemento	Descrição
Clientes	Agrupamento de clientes de acordo com as necessidades semelhantes.
Valor	De que forma nos podemos diferenciar da concorrência
Canais	Indicam a forma como pode ser feito o contacto com os clientes.
Relacionamento	Interação entre a empresa e o segmento do cliente escolhido.
Receitas	Representa a forma como é gerada receita com determinado nicho de clientes.
Recursos	Conjunto de pessoas e produtos que a empresa precisa para o processo funcionar.
Atividades	Conjunto de atividades a serem realizadas para que o modelo de negócio funcione.
Parceiros	Conjunto de parceiros que fazem o modelo de negócios funcionar.
Custos	Descreve quais os custos envolvidos no modelo de negócios.

6.2 Segmentos de clientes ou público-alvo

A aplicação pode aumentar o público de interesse visto que hoje em dia as pessoas como têm pouco tempo em casa para ver notícias ou outro tipo de conteúdo

de interesse, a aplicação vem preencher essa lacuna que existia, e explorar esse nicho de mercado.

O público-alvo pretendido são portugueses entre os 15 a 50 anos de idade, mais propriamente as mulheres visto que grande parte dos serviços disponibilizados está focado em produtos para mulheres.

O público-alvo está focado primeiramente nas idades mais novas pela aptidão no uso de novas tecnologias. O facto de a idade ser até aos 50 anos é ter verificado que as tecnologias não foram tão utilizadas nessas faixas etárias.

Em relação ao público mais focado ser as mulheres prende-se pelo simples fator de que estas estão mais preocupadas com a beleza e bem-estar do que propriamente os homens, embora hoje em dia os homens comessem cada vez mais a preocupar-se com a sua aparência.

6.3 Oferta de valor

A aplicação permite efetuar marcações *online* sem sair de casa, de forma a tornar mais cómoda uma marcação por parte de um utilizador.

Outra oferta de valor que poderia ser considerada seria a oferta de promoções semanais por parte dos profissionais de saúde/beleza, no entanto esta ideia ainda não foi colocada em prática.

6.4 Canais

A aplicação pode colocar anúncios diários de forma a cativar o cliente a usar a aplicação.

Poderiam também ser partilhadas fotos entre os utilizadores de forma a aumentar o interesse destes.

A empresa pode também colocar uns anúncios na sua página para despertar o interesse por parte do utilizador e redirecionando para a aplicação.

A Beauti tem a sua própria página do facebook, onde poderia colocar uma hiperligação para a aplicação.

6.5 Relacionamento

O cliente pode efetuar marcações em especialistas de saúde/beleza através da aplicação.

A aplicação fornece a possibilidade de efetuar marcações *online*.

O cliente pode usar a aplicação em qualquer lado desde que tenha acesso à internet.

6.6 Fontes de Receitas

Uma outra forma de despertar o interesse do utilizador é colocar a aplicação no Google Play Store, para que seja facilmente comercializada, desta forma o utilizador tem um acesso à aplicação mais confiável.

No sentido de estabelecer um modelo de aplicação lucrativa, serão cobrados ao profissional de saúde/beleza 0,50€ por marcação. Embora não seja cobrado nada ao utilizador por marcação nem pelo *download* da aplicação, esta alternativa de modelo de negócio mostra-se atrativa e rentável, isto porque, as aplicações gratuitas se forem bem exploradas são as que gerem mais dinheiro.

Uma outra possível fonte de receita seria vender a ideia inerente da aplicação a outras empresas interessadas neste tipo de funcionamento.

6.7 Recursos

O que é necessário para que o processo funcione é uma base de profissionais de beleza e saúde fidelizados.

Outro recurso necessário existir é existir uma base clientes que pretenda fazer marcações através do telemóvel.

6.8 Atividades chave

Para que o produto funcione, será feita a promoção através de colocação de anúncios no Facebook (https://www.facebook.com/beauti.pt?fref=ts) e no Web Site (http://www.beauti.pt).

Serão feitas também campanhas junto do cliente através da colocação de anúncios nos estabelecimentos dos profissionais.

Poderão também ser feitas propagandas na rádio, ou na televisão.

6.9 Parceiros chave

Os parceiros chave são o conjunto de profissionais de beleza ou saúde que sem eles a ideia não poderia avançar.

Outros parceiros podem ser outras empresas que pretendam aumentar colaborar com a empresa ou aplicação e queiram entrar no negócio como terceiros.

6.10 Custos

A estratégia de custo consistiu em diferentes fatores: abranger um público-alvo com menos capacidades financeiras e fornecer um valor razoável para marcação em um profissional de saúde/beleza.

O valor da aplicação no Google Store é gratuito. Assim é possível abranger todo o tipo de público-alvo.

Em relação aos descontos em profissionais de saúde/beleza, esses descontos são definidos pelos próprios profissionais.

7 Testes de usabilidade

Segundo Elizabeth Snowdon [Elizabeth Snowdon, 2010] a usabilidade pode poupar tempo e dinheiro e determinar a quota de mercado de uma applicação. A usabilidade incorpora o feedback de um utilizador direto em todo o ciclo de desenvolvimento, a fim de reduzir custos e criar produtos e ferramentas que atendam às necessidades do utilizador.

7.1 Métricas de usabilidade – Norma ISO 9241-11

Segundo Justin Mifsud [Justin Mifsud, 2015], a norma ISO 9241-11 define usabilidade como "a medida em que um produto pode ser usado por utilizadores para alcançar objetivos específicos com eficácia, eficiência e satisfação num determinado contexto de utilização".

Eficácia: A precisão e a plenitude com a qual os utilizadores alcançaram metas específicas.

Eficiência: Os recursos gastos em relação à precisão e plenitude com as quais os utilizadores atingem metas.

Satisfação: O conforto e a aceitabilidade de uso.

Para o cálculo destas três métricas foi necessário realizar um questionário a diversos utilizadores. A tabela 16 ilustra as respostas ao questionário de satisfação realizado a 5 utilizadores iniciantes, como também ilustra o tempo que o utilizador gastou a realizar determinada tarefa.

Tabela 16 – Tarefas realizadas com sucesso/tempo de realização da tarefa

	Tarefa 1	**Tarefa 2**	**Tarefa 3**	**Tarefa 4**	**Tarefa 5**
	Conseguiu fazer login na aplicação?	**Percebeu para que servia a pesquisa?**	**Soube onde alterar os dados do perfil?**	**Teve facilidade a fazer uma marcação?**	**Percebeu onde veria a marcação efetuada?**
Utilizador 1	sim/2s	não/4s	sim/3s	não/4s	sim/4s
Utilizador 2	sim/2s	não/4s	não/5s	não/3s	sim/3s
Utilizador 3	sim/3s	sim/3s	sim/3s	não/5s	não/6s
Utilizador 4	não/4s	sim/3s	sim/4s	sim/6s	sim/3s
Utilizador 5	sim/2s	sim/2s	não/4s	sim/4s	sim/4s

7.1.1 Eficicácia

A eficácia é calculada pela razão entre as tarefas concluídas com sucesso e o número total de tentativas de as executar. É atribuído um valor binário de '1' se o participante no teste consegue completar uma tarefa e '0' se ele não consegue.

Para calcular a eficácia é aplicada a seguinte equação:

$$\text{Eficácia} = \frac{Nº\ de\ tarefas\ realizadas\ com\ sucesso}{Nº\ de\ tarefas\ totais\ realizadas} * 100\% \qquad (2)$$

Exemplo: Cálculo da eficácia para a tarefa 1

No exemplo anterior temos: 5 utilizadores a executar uma tarefa utilizando o mesmo sistema. No fim da sessão de teste apenas 4 utilizadores conseguem alcançar o objectivo da tarefa 1. Utilizando a equação acima, a eficácia global do utilizador do sistema é a que se segue:

Nº de tarefas realizadas com sucesso = 4 (nº de vezes que o utilizador conseguiu realizar a tarefa).
Nº de tarefas totais realizadas = 5 (nº de utilizadores).

Eficicácia= $\frac{4}{5} * 100\% = 80\%$

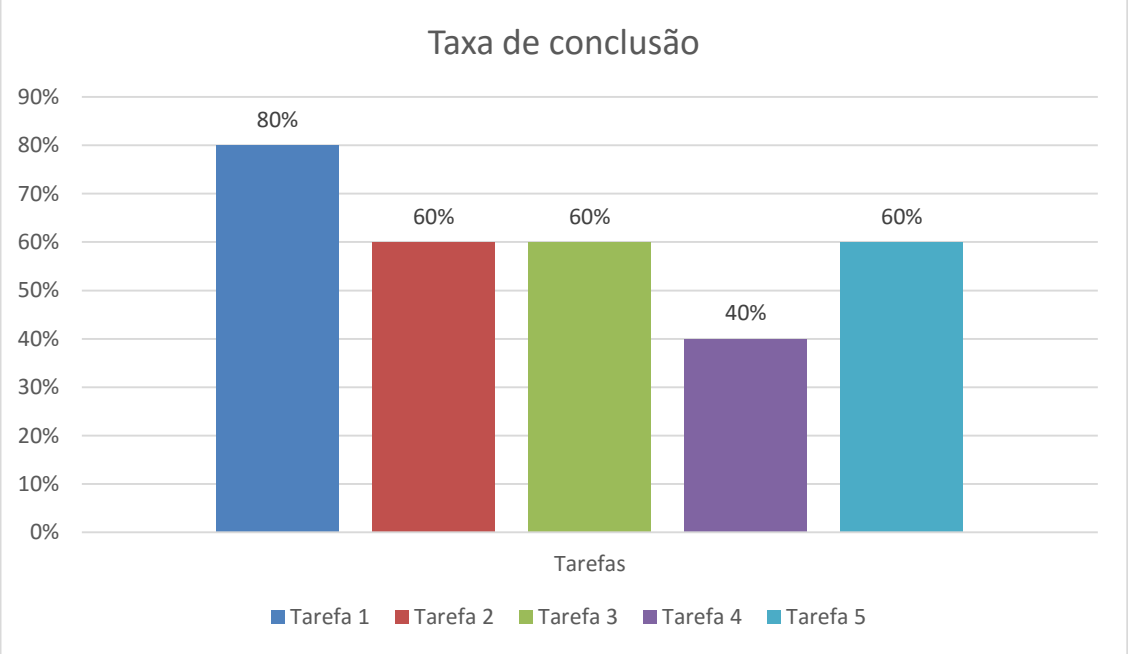


Figura 33 – Gráfico com as tarefas dadas ao utilizador.

Através da análise à tabela 16 obtemos o gráfico ilustrado na figura 33 em que a taxa de conclusão indica a percentagem de sucesso na execução de uma tarefa.

7.1.2 Eficiência

A **eficiência baseada em tempo** é a velocidade de trabalho (segundos e/ou minutos) com que um determinado utilizador executa uma tarefa com sucesso. A eficiência geral com base no tempo de um produto é dada pela seguinte fórmula.

$$\text{Eficiência baseada em tempo} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} nij/tij}{NR} \quad (3)$$

N = O número total de tarefas
R = O número de utilizadores
n_{ij} = O resultado da tarefa i pelo utilizador j; se o usuário for concluída com êxito a tarefa, em seguida, Nij = 1, se não, então Nij = 0
t_{ij} = O tempo gasto pelo utilizador j para completar a tarefa i. Se a tarefa não for concluída, então o tempo é medido até o momento em que o utilizador encerra a tarefa.

Exemplo: Cálculo de eficiência baseada em tempo para a tarefa 1

Mais uma vez, vamos dar um exemplo prático. Existem 5 utilizadores que usam o mesmo produto para tentar executar a mesma tarefa (1 tarefa). 4utilizadores conseguem concluí-la com êxito - tendo 2, 2, 3, 4 e 2 segundos respectivamente. O quarto utilizador leva 4 segundos e não consegue concluir a tarefa 1.

N = O número total de tarefas = 1
R = O número de utilizadores = 5
Utilizador 1: Nij = 1 e Tij = 2s
Utilizador 2: Nij = 1 e Tij = 2s
Utilizador 3: Nij = 1 e Tij = 3s
Utilizador 4: Nij = 0 e Tij = 4s
Utilizador 5: Nij = 1 e Tij = 2s

$$\text{Eficiência baseada em tempo} = \frac{(1/2+1/2+1/3+0/4+1/2)}{1*5} = 0{,}36 \text{ objetivos/seg}$$

A **eficiência relativa global** utiliza a relação entre o tempo gasto pelos utilizadores que completaram com sucesso a tarefa em relação ao tempo total de todos os utilizadores.

$$\text{Eficiência relativa global} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} nij\ tij}{\sum_{j=1}^{R}\sum_{i=1}^{N} tij} * 100\% \quad (4)$$

Exemplo: Cálculo da eficiência relativa global para a tarefa 1

N = O número total de tarefas = 1
R = O número de utilizadores = 5
Utilizador 1: Nij = 1 e Tij = 2s
Utilizador 2: Nij = 1 e Tij = 2s
Utilizador 3: Nij = 1 e Tij = 3s
Utilizador 4: Nij = 0 e Tij = 4s
Utilizador 5: Nij = 1 e Tij = 2s

Eficiência relativa global = $\left(\frac{(1*1)+(1*2)+(1*3)+(0*4)+(1*2)}{(1+2+3+4+2)}\right) * 100\% = 66{,}66\%$

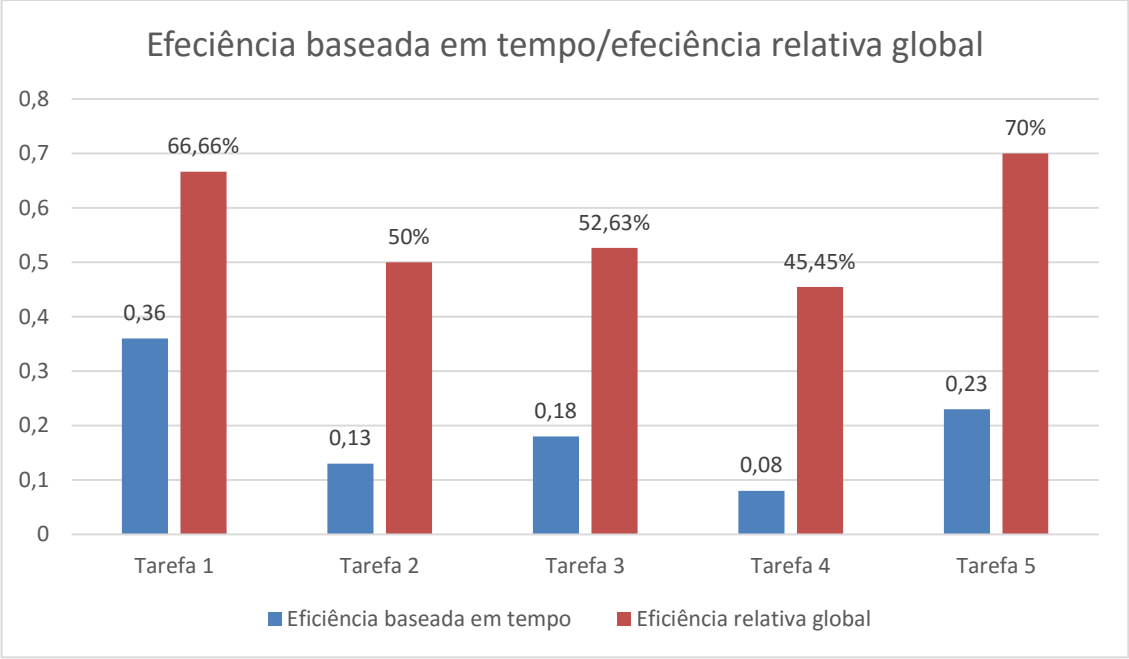


Figura 34 – Eficiência

A figura 34 representa a efeciência na realização de uma tarefa por um utilizador. A barra azul é a eficiência baseada em tempo, esta é dada pelo número de objetivos com que os utilizadores realizam uma tarefa num determinado tempo (segundos). A barra vermelha é a eficiência relativa global, que dá a percentagem de utilizadores que realizaram a tarefa.

7.1.3 Satisfação

O nível de satisfação é medido através de questionários de satisfação realizados após a execução das tarefas, esses questionários são dados a fim de medir o quão difícil essa tarefa era. Este tipo de questionário é composto normalmente por cinco questões e tomam a forma de escala de *rating*. O que será usado é o ASQ (questionário realizado

após a tarefa), composto por 3 níveis de questões (fácil, moderada, difícil) e realizado após o cenário. A tabela 17 ilustra a escala de dificuldade de cada tarefa por diferente utilizador.

Tabela 17 – Níveis de dificuldade das tarefas

	Tarefa 1	**Tarefa 2**	**Tarefa 3**	**Tarefa 4**	**Tarefa 5**
Utilizador 1	Fácil	Moderada	Fácil	Moderada	Fácil
Utilizador 2	Fácil	Moderada	Moderada	Moderada	Fácil
Utilizador 3	Fácil	Fácil	Fácil	Moderada	Moderada
Utilizador 4	Moderada	Fácil	Fácil	Fácil	Fácil
Utilizador 5	Fácil	Fácil	Moderada	Fácil	Fácil

7.2 Conclusão

Através desde capítulo ficou provado que com a utilização de métricas de usabilidade, é possível observar e quantificar a usabilidade de qualquer tipo de sistema (software, hardware, web ou uma aplicação móvel). Isso ocorre porque as métricas são baseados em extensas pesquisa e testes feitos por vários acadêmicos e especialistas e têm resistido ao longo do tempo. Além disso estas métricas cobrem os três elementos fundamentais de usabilidade (eficácia, eficiência e satisfação), garantindo uma quantificação do sistema que está sendo testado.

Outra conclusão retirada foi que que um utilizador mesmo iniciante não tem dificuldades na utilização do sistema e consegue perceber a utilidade do mesmo.

8 Conclusões e trabalho futuro

Neste capítulo são apresentadas algumas conclusões através da análise de resultados mostrando resumidamente aquilo que foi desenvolvido ao longo do trabalho.

São também apresentadas algumas dificuldades encontradas no trabalho, e por último possíveis melhorias a implementar no mesmo.

8.1 Análise de resultados

Após feita a análise ao REST, pode-se concluir que este é bem melhor de usar que o SOAP devido não apenas à sua facilidade de utilização como também à liberdade de escolha na linguagem de mensagem (XML ou JSON). O SOAP também se apresenta como solução válida quando é necessário que um sistema seja mais seguro, como por exemplo, bancos ou outro tipo de sistema. No entanto a utilização de REST em conjunto com outras tecnologias permite igualar os níveis de seguranção do SOAP.

Em relação aos Web Services, estes são uma boa solução para integrar sistemas operativos diferentes (iOS, Android, Windows). E qualquer uma das arquiteturas estudadas, REST e SOAP, são uma possível alternativa para quem deseja usar um sistema deste género.

Conclui-se também que surgem desafios na área de aplicações móveis para os programadores, tais como os tamanhos dos ecrãs, ou o limite de memória do dispositivo. Estes são aspetos a ter em conta no desenvolvimento de uma aplicação de forma a não comprometer a mesma. Neste sentido foram feitos vários testes aos layouts, e usado o menor espaço de memória possível de forma a não comprometer o dispositivo do utilizador.

Quanto ao estudo realizado foram atingidos os objetivos deliniados. Neste estudo era pretendido fazer comparações entre as diferentes arquiteturas REST e SOAP, e esse objetivo foi conseguido no capítulo dos testes, sendo analisados as diferentes arquiteturas para Web Services.

Ao ser desenvolvido este trabalho deparei-me com algumas dificuldades, nomeadamente no âmbito do que era pretendido para esta tese de mestrado.

A familiarização com a linguagem Android foi uma adversidade que inicialmente, dificultou o trabalho, mas que foi possível ultrapassar através de várias fontes online.

A própria comparação entre o protocolo SOAP e arquitetura REST não é muito precisa, visto que os tempos de resposta que vão desde a submissão de um pedido até à obtenção da resposta variam.

8.2 Trabalho Futuro

Existem algumas melhorias que poderiam ser introduzidas neste trabalho, como por exemplo a aplicação Beauti apesar de considerar que esta será uma grande ajuda para os centros de beleza e não só, lamentavelmente esta apenas serviu para testar o funcionamento de uma aplicação baseada no protocolo REST no Android. Teria sido interessante criar uma aplicação funcional que estivesse no mercado.

Podiam também ter sido usados os Web Services da Beauti, mas devido a requisitos de confidencialidade os testes apenas foram feitos a Web Services externos.

Por fim o trabalho poderia ter sido extendido a mais tecnologias de acesso a Web Services, assim a análise teria muito maior interesse e detalhe sobre qual seria a tecnologia mais adequada a se usar em cada caso.

Referências

[Joaquim Anacleto, 2012]	Joaquim Anacleto – Desenvolvimento de uma aplicação web para dispositivos móveis. Disponível em: http://wiki.di.uminho.pt/twiki/pub/Research/APEX/Publications/tese.pdf [Acedido a 08/09/2014]
[Joseph Bih, 2006]	Joseph Bih – Service Oriented Architecture a New Paradigm to Implement Dynamic E-Business Solutions. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.4303&rep=rep1&type=pdf (p. 2 – 3) [Acedido a 10/04/2015]
[David Gassner, 2015]	David Gassner – What is a web service? Disponível em: http://www.lynda.com/OData-tutorials/What-web-service/126131/145941-4.html [Acedido a 20/04/2015]
[K. Sahin, 2008]	K. Sahin, Gumusay – Service Oriented Architecture (SOA) based Web Services for geographic information systems. Disponível em: http://www.isprs.org/proceedings/XXXVII/congress/2_pdf/5_WG-II-5/03.pdf [Acedido a 11/04/2015]
[tuturialspoint, 2015a]	Tuturialspoint – What are Web Services? Disponível em: http://www.tutorialspoint.com/webservices/what_are_web_services.htm [Acedido a 12/04/2015]
[w3schools, 2015],	w3schools – SOAP HTTP Binding Disponível em: http://www.w3schools.com/webservices/ws_soap_httpbinding.asp [Acedido a 20/08/2015]
[W3C, 2007]	W3C – Using Various Protocol Bindings Disponível em: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/#transport [Acedido a 20/08/2015]
[tuturialspoint, 2015b]	Tuturialspoint – SOAP Disponível em: http://www.tutorialspoint.com/soap/index.htm [Acedido a 05/09/2014]
[Steve Francia, 2012]	Steve Francia – REST vs SOAP. Disponível em: http://spf13.com/post/soap-vs-rest [Acedido a 02/10/2014]
[Heather Kreger, 2001]	Heather Kreger – Web Services Conceptual Architecture. Disponível em: http://www.cs.uoi.gr/~pvassil/downloads/WebServices/Tutorials/WebServicesConceptualArchitecture.pdf [Acedido a 22/04/2015]
[Paulo Granja, 2014]	"REST Web Services", Sistemas Distribuídos aula 4, 5 – Paulo Granja
[Stefan Tilkov, 2007]	Stefan Tilkov – A Brief Introduction to REST Disponível em: http://www.infoq.com/articles/rest-introduction [Acedido a 20/08/2015]
[André Gomes, 2008]	André Faria Gomes – Uma rápida Introdução ao REST Disponível em: http://www.infoq.com/br/articles/rest-introduction [Acedido a 17/04/2015]

[Mike Rozlog, 2010]	Mike Rozlog – REST e SOAP quando devo usar cada um? Ou ambos? Disponível em: http://www.infoq.com/articles/rest-soap-when-to-use-each [Acedido a 14/08/2014]
[Jagadeesh Motamarri, 2013]	Jagadeesh Motamarri – Compare Restful vs SOAP Web Services. Disponível em: http://java.dzone.com/articles/j2ee-compare-restful-vs-soap [Acedido a 13/08/2014]
[Sonoo Jaiswal, 2014]	Sonoo Jaiswal – SOAP vs REST Web Services. Disponível em: http://www.javatpoint.com/soap-vs-rest-web-services [Acedido a 13/08/2014]
[Nordic, 2015]	Nordic – REST vs SOAP. Disponível em: http://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/ [Acedido a 23/04/2015]
[Packetizer, 2014]	Packetizer, Inc. – REST. Disponível em: http://www.packetizer.com/ws/rest.html [Acedido a 13/10/2014]
[Kishor Wagh, 2012]	Kishor Wagh, Dr. Ravindra Thool – A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. Disponível em: http://iiste.org/Journals/index.php/JIEA/article/viewFile/2063/2042 [Acedido a 13/10/2014]
[Elkstein, 2008]	Elkstein. Aprender REST: Um tutorial. Disponível em: http://rest.elkstein.org/2008/02/what-is-rest.html [Acedido a 21/08/2014]
[REST Api tutorial, 2015]	Rest Api tutorial – "What Is REST". Disponível em: http://www.restapitutorial.com/lessons/whatisrest.html [Acedido a 23/08/2014]
[Maurizio Marchese, 2014]	Maurizio Marchese – SOAP. Disponível em: https://www.youtube.com/watch?v=l2QrCqzRhWo [Acedido a 26/08/2014]
[Alexandre Macedo, 2010]	Alexandre Macedo – WebService Restful utilizando JSON. Disponível em: http://www.k19.com.br/artigos/webservice-restful-utilizando-json/
[Nicholas Quaine, 2007]	Tutorialspoint – O que é SOAP? Disponível em: http://www.soapuser.com/basics1.html [Acedido a 26/08/2014]
[Microsoft, 2014]	Microsoft – Especificações de Mensagem. Disponível em: http://msdn.microsoft.com/en-us/library/ms951268.aspx [Acedido a 27/08/2014]
[W3C - Steve Bratt, 2004]	W3C - Steve Bratt – Exemplo de mensagem SOAP. Disponível em: http://www.w3.org/2004/06/03-google-soap-wsdl.html [Acedido a 27/08/2014]
[Jakob Jenkov, 2015a]	Jakob Jenkov – "SOAP". Disponível em: http://tutorials.jenkov.com/soap/index.html [Acedido a 27/08/2014]
[Jakob Jenkov, 2015b]	Jakob Jenkov – Web Service Message Formats. Disponível em: http://tutorials.jenkov.com/web-services/message-formats.html#soap [Acedido a 27/05/2015]
[IBM, 2014]	IBM – "The SOAP header". Disponível em: http://pic.dhe.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=%2Fcom.ibm.cics.ts.webservices.doc%2Fconcepts%2Fsoap%2Fdfhws_header.html [Acedido a 20/08/2014]

[Michael P. Papazoglou, 2008]	Michael P. Papazoglou – Web Services. Disponível em: http://www.cs.colorado.edu/~kena/classes/7818/f08/lectures/lecture_3_soap.pdf [Acedido a 04/09/2014]
[John Mueller, 2013]	John Mueller – "Understanding SOAP and REST Basics. Disponível em: http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/ [Acedido a 20/09/2014]
[Barry Harmsen, 2014]	Barry Harmsen – "QlikView hash functions and collisions". Disponível em: http://www.qlikfix.com/2014/03/11/hash-functions-collisions/ [Acedido a 01/10/2014]
[Keshav Bhatia, 2010]	Keshav Bhatia – "Marketing Mix". Disponível em: http://pt.slideshare.net/keshavbhatia/marketing-mix-2976624 [Acedido a 16/10/2014]
[Vitor M., 2011]	Vitor M. – 99% dos Androids têm fuga de dados confidenciais. Disponível em: http://pplware.sapo.pt/informacao/99-dos-androids-tem-fuga-de-dados-confidenciais/ [Acedido a 18/09/2014]
[Pedro Pinto, 2010]	Pedro Pinto – Criptografia simétrica e assimétrica. Disponível em: http://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/ [Acedido a 18/09/2014]
[Karan Balkar, 2013]	Karan Balkar – Implement SAH1 and MD5 hashing in Android. Disponível em: http://karanbalkar.com/2013/05/tutorial-28-implement-sha1-and-md5-hashing-in-android/ [Acedido a 18/09/2014]
[Daniel Flores Bastos, 2011]	Daniel Flores Bastos – O que é MVC Disponível em: http://www.oficinadanet.com.br/artigo/desenvolvimento/o_que_e_model-view-controller_mvc [Acedido a 06/10/2014]
[Joshua Bloch, 2013]	Joshua Bloch – "hashing". http://www.cs.princeton.edu/~rs/AlgsDS07/10Hashing.pdf [Acedido a 1/10/2014]
[LinkedIn, 2015]	LinkedIn – Getting started with the REST API. Disponível em: https://developer.linkedin.com/docs/rest-api [Acedido a 28/04/2015]
[Joe Betz, 2013]	Joe Betz – Rest.li: RESTful Service Architecture at Scale Disponível em: https://engineering.linkedin.com/architecture/restli-restful-service-architecture-scale [Acedido a 28/04/2015]
[Twitter, 2015]	Twitter – GET Status Disponível em: https://dev.twitter.com/rest/reference/get/statuses/user_timeline [Acedido a 28/04/2015]
[Caelum, 2015]	Caelum – REST Disponível em: http://www.infoq.com/br/articles/rest-introduction [Acedido a 30/04/2015]
[Jean Lima, 2012]	Jean Lima – WEB SERVICES (SOAP X REST) Disponível em: http://www.fatecsp.br/dti/tcc/tcc00056.pdf [Acedido a 03/05/2015]
[Mkuchtiak, 2006]	Mkuchtiak – Web Services: REST vs. SOAP Disponível em: https://blogs.oracle.com/milan/entry/web_services_rest_vs_soap [Acedido a 03/05/2015]
[M. Vaqqas, 2014]	M. Vaqqas – RESTful Web Services: A Tutorial Disponível em: http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069

	[Acedido a 03/05/2015]
[twitter, 2015b]	Twitter – Documentation Disponível em: https://dev.twitter.com/streaming/overview [Acedido a 11/05/2015]
[Slack, 2015]	Slack Disponível em: https://api.slack.com/ [Acedido a 14/05/2015]
[Cristiano Caetano, 2015].	Cristiano Caetano – Testes de Web Services Disponível em: http://www.linhadecodigo.com.br/artigo/1286/soapui-testes-de-web-services-rapido-e-descomplicado.aspx [Acedido a 30/07/2015]
[SmartBear Software, 2015a]	SmartBear Software – Working with WSDLs Disponível em: http://www.soapui.org/soap-and-wsdl/working-with-wsdls.html [Acedido a 30/07/2015]
[SmartBear Software, 2015b]	SmartBear Software – Getting Started with REST Testing Disponível em: http://www.soapui.org/rest-testing/getting-started.html [Acedido a 30/07/2015]
[SmartBear Software, 2015c]	SmartBear Software – Creating and Running LoadTests Disponível em: http://www.soapui.org/load-testing/creating-and-running-loadtests.html http://www.soapui.org/load-testing/creating-and-running-loadtests.html
[Thanachart, 2010]	Thanachart Numnonda – Introduction to SOAP Disponível em: http://www.slideshare.net/imcinstitute/java-web-services-25-introduction-to-soap?from_action=save [Acedido a 30/07/2015]
[Ernesto Damiani, 2015]	Ernesto Damiani – SOAP message structure Disponível em: http://ra.crema.unimi.it/turing/materiale/admin/corsi/SOA/materiale/SOA3.pdf [Acedido a 30/07/2015]
[Elizabeth Snowdon, 2015]	Elizabeth Snowdon – Testes de usabilidade Disponível em: http://pt.slideshare.net/esnowdon/usability-testing-101-an-introduction [Acedido a 14/08/2015]
[Justin Mifsud, 2015]	Justin Mifsud – Métricas de usabilidade Disponível em: http://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability/ [Acedido a 14/08/2015]
[W3C, 2004]	W3C – Web Services Disponível em: http://www.w3.org/TR/ws-gloss/ [Acedido a 19/08/2015]
[Sandro Santos, 2014]	Sandro Santos – Canvas de modelo de negócios Disponível em: http://sandrossantos.com/canvas-modelo-negocio/ [Acedido a 01/08/2015]
[Alexander Osterwalder, 2009]	Alexander Osterwalder Disponível em: http://www.businessmodelgeneration.com/downloads/businessmodelgeneration_preview.pdf [Acedido a 01/08/2015]

9 Anexo A

Tabela 18 – SOAP vs REST (XML): tempos de resposta

SOAP Tamanho dos dados (em bytes): 382		**REST (XML)** Tamanho dos dados (em bytes): 1176	
Tempo de Resposta (ms)		**Tempo de Resposta (ms)**	
386	360	66	113
413	404	79	81
431	332	90	80
367	357	77	77
325	359	88	82
395	350	84	81
419	368	81	80
348	324	83	66
357	358	81	87
327	330	83	90
317	397	80	92
442	360	83	85
395	375	65	64
365	373	85	151
364	371	73	99
332	328	80	81
365	375	81	80
467	329	82	81
360	364	79	77
328	334	86	80

Fonte SOAP: http://www.webservicex.com/CurrencyConvertor.asmx?wsdl

Fonte REST: http://maps.googleapis.com/maps/api/geocode/xml?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false

Testes REST vs SOAP

Nas figuras seguintes são apresentados alguns dos resultados dos testes efetuados na tabela anterior.

Figura 35 – REST, teste 1

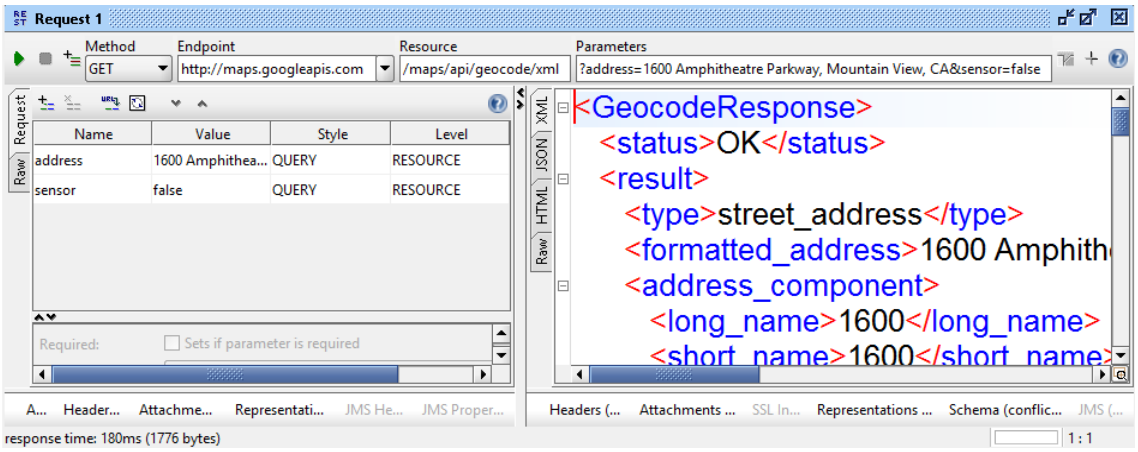


Figura 36 – REST, teste 2

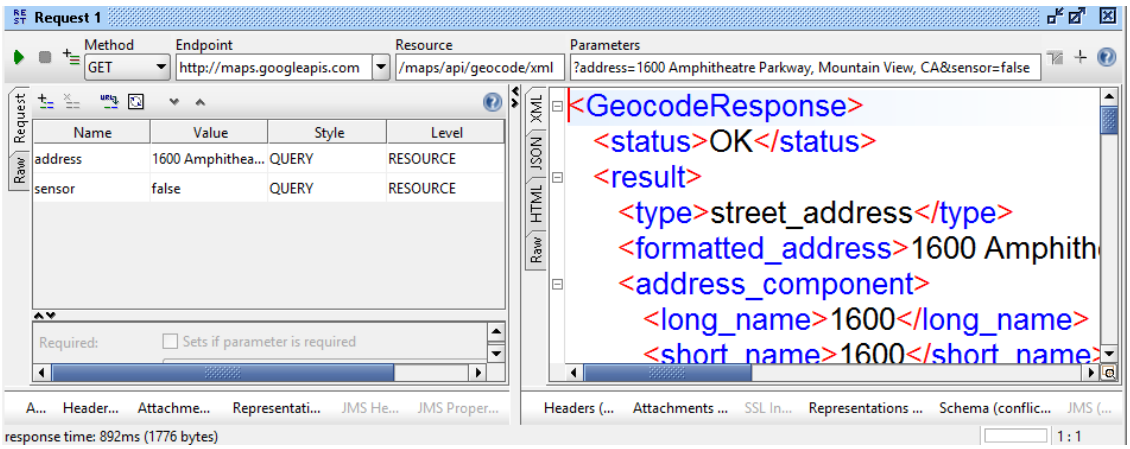


Figura 37 – REST, teste 3

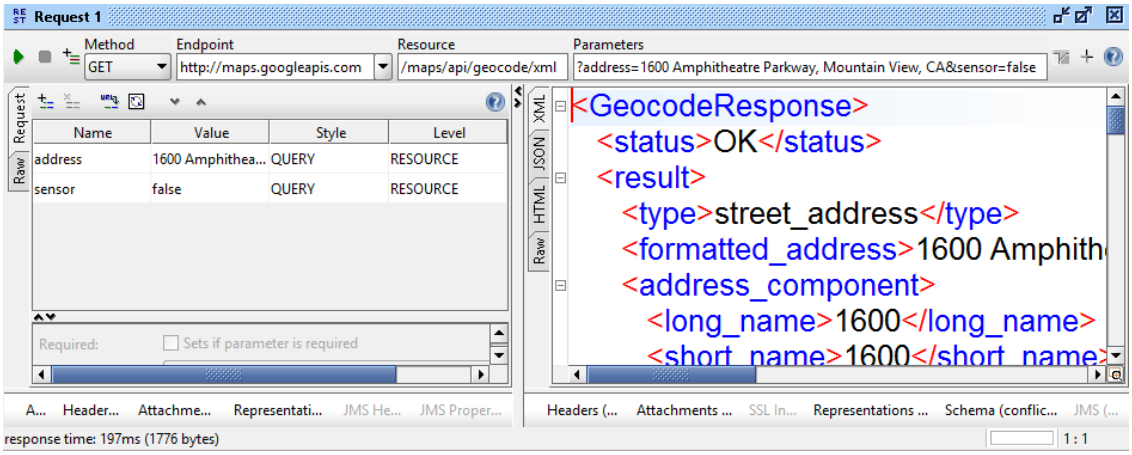


Figura 38 – REST, teste 4

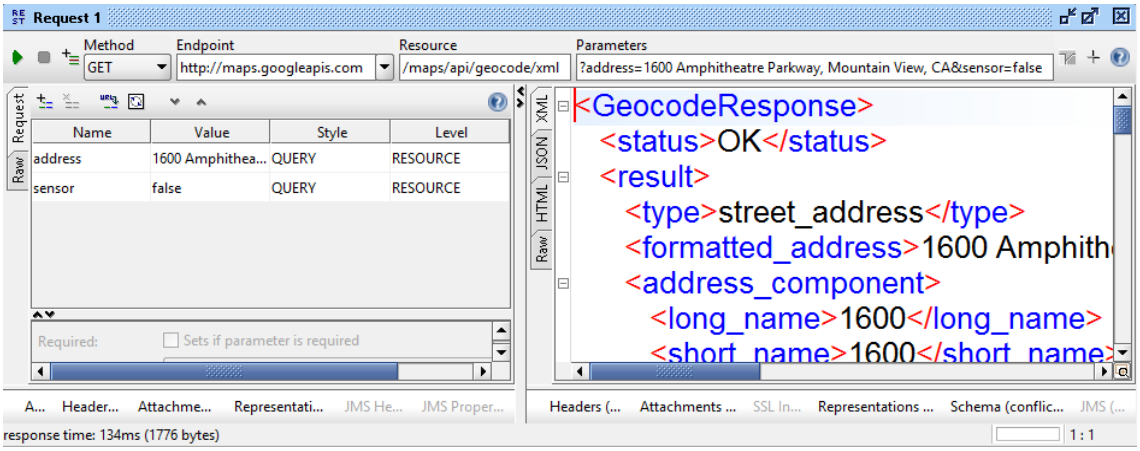


Figura 39 – SOAP, teste 1

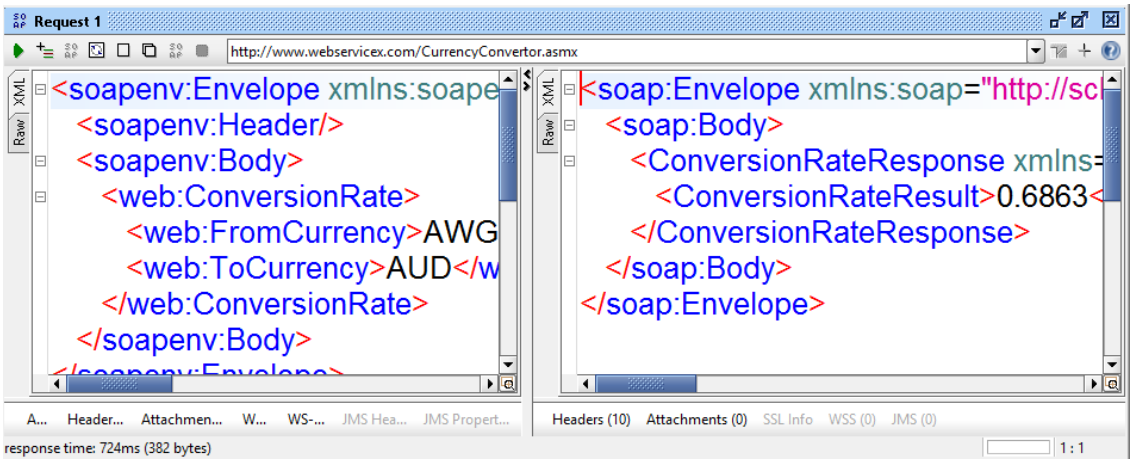


Figura 40 – SOAP, teste 2

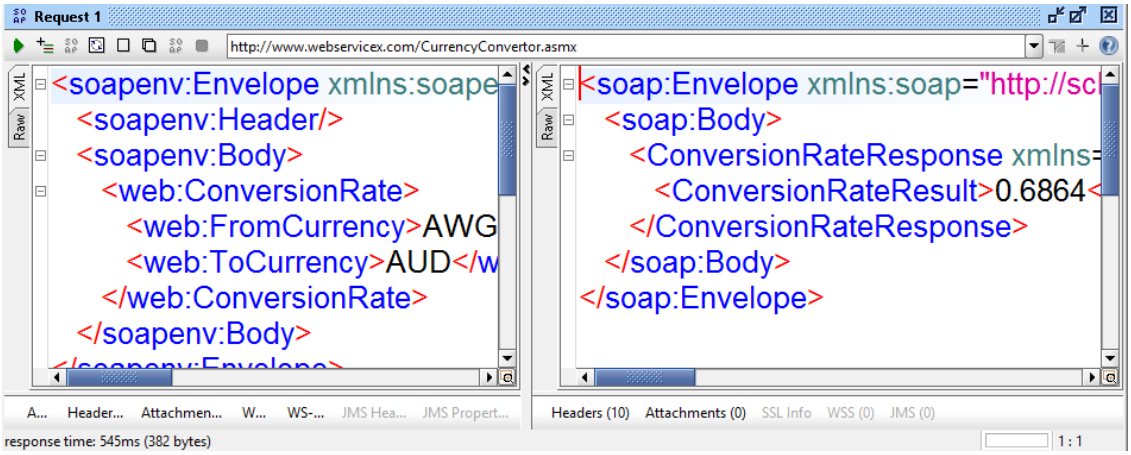


Figura 41 – SOAP, teste 3

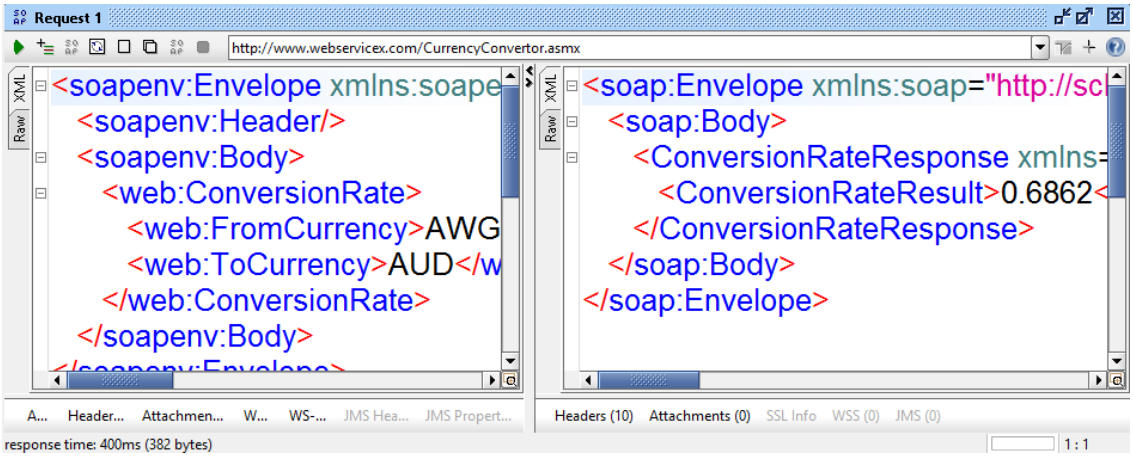
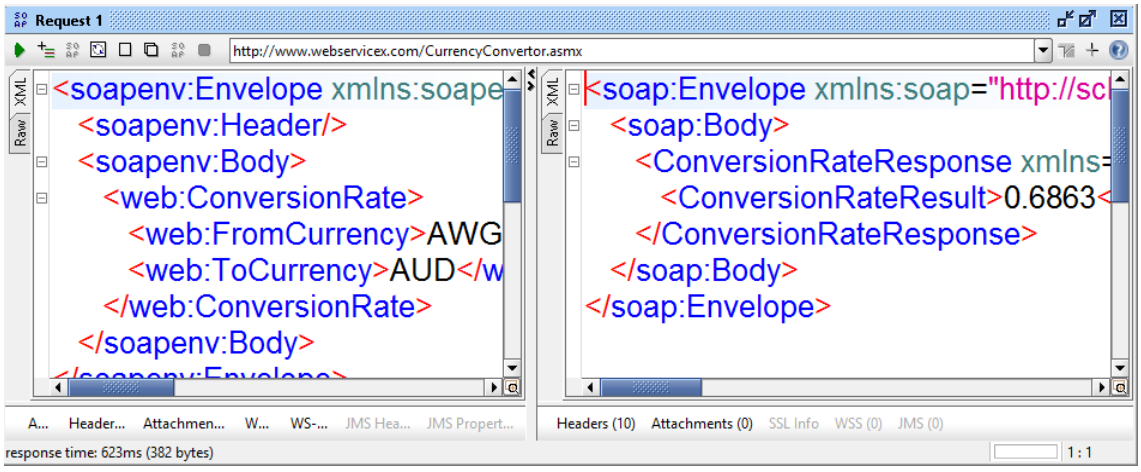


Figura 42 – SOAP, teste 4



10 Anexo B

Tabela 19 – REST (XML) vs REST (JSON): Tempos de resposta e tamanho de dados.

REST (XML) Tamanho dos dados (em bytes): 1176		**REST (JSON) – método GET** Tamanho dos dados (em bytes): 1179	
Tempo de Resposta (ms)		**Tempo de Resposta (ms)**	
66	113	72	60
79	81	66	60
90	80	74	58
77	77	61	65
88	82	59	58
84	81	63	60
81	80	58	55
83	66	62	60
81	87	60	55
83	90	58	60
80	92	79	60
83	85	68	56
65	64	63	59
85	151	62	55
73	99	61	59
80	81	81	63
81	80	56	115
82	81	58	57
79	77	60	61
86	80	70	65

Fonte REST em XML: http://maps.googleapis.com/maps/api/geocode/xml?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false

Fonte REST em JASON: http://maps.googleapis.com/maps/api/geocode/json?address=Rio&sensor=false

11 Anexo C

Exemplo de um pedido GET da Twitter API


```
[
  {
    "coordinates": null,
    "favorited": false,
    "truncated": false,
    "created_at": "Wed Aug 29 17:12:58 +0000 2012",
    "id_str": "240859602684612608",
    "entities": {
      "urls": [
        {
          "expanded_url":  "https://dev.twitter.com/blog/twitter-certified-
products",
          "url": "https://t.co/MjJ8xAnT",
          "indices": [
            52,
            73
          ],
          "display_url": "dev.twitter.com/blog/twitter-c\u2026"
        }
      ],
      "hashtags": [

      ],
      "user_mentions": [

      ]
    },
    "in_reply_to_user_id_str": null,
    "contributors": null,
    "text":  "Introducing  the  Twitter  Certified  Products  Program:
https://t.co/MjJ8xAnT",
    "retweet_count": 121,
    "in_reply_to_status_id_str": null,
    "id": 240859602684612608,
    "geo": null,
    "retweeted": false,
    "possibly_sensitive": false,
    "in_reply_to_user_id": null,
    "place": null,
    "user": {
      "profile_sidebar_fill_color": "DDEEF6",
      "profile_sidebar_border_color": "C0DEED",
      "profile_background_tile": false,
      "name": "Twitter API",
      "profile_image_url":
"http://a0.twimg.com/profile_images/2284174872/7df3h38zabcvjylnyfe3_normal.
png",
      "created_at": "Wed May 23 06:01:13 +0000 2007",
      "location": "San Francisco, CA",
```

```
      "follow_request_sent": false,
      "profile_link_color": "0084B4",
      "is_translator": false,
      "id_str": "6253282",
      "entities": {
        "url": {
          "urls": [
            {
              "expanded_url": null,
              "url": "http://dev.twitter.com",
              "indices": [
                0,
                22
              ]
            }
          ]
        },
        "description": {
          "urls": [

          ]
        }
      },
      "default_profile": true,
      "contributors_enabled": true,
      "favourites_count": 24,
      "url": "http://dev.twitter.com",
      "profile_image_url_https":
"https://si0.twimg.com/profile_images/2284174872/7df3h38zabcvjylnyfe3_norma
l.png",
      "utc_offset": -28800,
      "id": 6253282,
      "profile_use_background_image": true,
      "listed_count": 10775,
      "profile_text_color": "333333",
      "lang": "en",
      "followers_count": 1212864,
      "protected": false,
      "notifications": null,
      "profile_background_image_url_https":
"https://si0.twimg.com/images/themes/theme1/bg.png",
      "profile_background_color": "C0DEED",
      "verified": true,
      "geo_enabled": true,
      "time_zone": "Pacific Time (US & Canada)",
      "description": "The Real Twitter API. I tweet about API changes,
service issues and happily answer questions about Twitter and our API. Don't
get an answer? It's on my website.",
      "default_profile_image": false,
      "profile_background_image_url":
"http://a0.twimg.com/images/themes/theme1/bg.png",
      "statuses_count": 3333,
      "friends_count": 31,
      "following": null,
      "show_all_inline_media": false,
      "screen_name": "twitterapi"
    },
    "in_reply_to_screen_name": null,
    "source":       "<a       href="//sites.google.com/site/yorufukurou/\""
rel="\"nofollow\"">YoruFukurou</a>",
```

```
    "in_reply_to_status_id": null
  },
  {
    "coordinates": null,
    "favorited": false,
    "truncated": false,
    "created_at": "Sat Aug 25 17:26:51 +0000 2012",
    "id_str": "239413543487819778",
    "entities": {
      "urls": [
        {
          "expanded_url": "https://dev.twitter.com/issues/485",
          "url": "https://t.co/p5bOzH0k",
          "indices": [
            97,
            118
          ],
          "display_url": "dev.twitter.com/issues/485"
        }
      ],
      "hashtags": [

      ],
      "user_mentions": [

      ]
    },
    "in_reply_to_user_id_str": null,
    "contributors": null,
    "text": "We are working to resolve issues with application management &
logging in to the dev portal: https://t.co/p5bOzH0k ^TS",
    "retweet_count": 105,
    "in_reply_to_status_id_str": null,
    "id": 239413543487819778,
    "geo": null,
    "retweeted": false,
    "possibly_sensitive": false,
    "in_reply_to_user_id": null,
    "place": null,
    "user": {
      "profile_sidebar_fill_color": "DDEEF6",
      "profile_sidebar_border_color": "C0DEED",
      "profile_background_tile": false,
      "name": "Twitter API",
      "profile_image_url":
"http://a0.twimg.com/profile_images/2284174872/7df3h38zabcvjylnyfe3_normal.
png",
      "created_at": "Wed May 23 06:01:13 +0000 2007",
      "location": "San Francisco, CA",
      "follow_request_sent": false,
      "profile_link_color": "0084B4",
      "is_translator": false,
      "id_str": "6253282",
      "entities": {
        "url": {
          "urls": [
            {
              "expanded_url": null,
              "url": "http://dev.twitter.com",
```

```
            "indices": [
              0,
              22
            ]
          }
        ]
      },
      "description": {
        "urls": [

        ]
      }
    },
    "default_profile": true,
    "contributors_enabled": true,
    "favourites_count": 24,
    "url": "http://dev.twitter.com",
    "profile_image_url_https":
"https://si0.twimg.com/profile_images/2284174872/7df3h38zabcvjylnyfe3_norma
l.png",
    "utc_offset": -28800,
    "id": 6253282,
    "profile_use_background_image": true,
    "listed_count": 10775,
    "profile_text_color": "333333",
    "lang": "en",
    "followers_count": 1212864,
    "protected": false,
    "notifications": null,
    "profile_background_image_url_https":
"https://si0.twimg.com/images/themes/theme1/bg.png",
    "profile_background_color": "C0DEED",
    "verified": true,
    "geo_enabled": true,
    "time_zone": "Pacific Time (US & Canada)",
    "description": "The Real Twitter API. I tweet about API changes,
service issues and happily answer questions about Twitter and our API. Don't
get an answer? It's on my website.",
    "default_profile_image": false,
    "profile_background_image_url":
"http://a0.twimg.com/images/themes/theme1/bg.png",
    "statuses_count": 3333,
    "friends_count": 31,
    "following": null,
    "show_all_inline_media": false,
    "screen_name": "twitterapi"
  },
  "in_reply_to_screen_name": null,
  "source":       "<a       href="//sites.google.com/site/yorufukurou/\""
rel="\"nofollow\"">YoruFukurou</a>",
  "in_reply_to_status_id": null
 }
]
```


Código 18 – Resposta em JSON [Twitter, 2015].