
**RODRIGO FERREIRA DOS SANTOS GONÇALVES**

# DESENVOLVIMENTO DE *WEB SERVICE* PARA APLICATIVO BASEADO EM LOCALIZAÇÃO GEOGRÁFICA APLICADO À MOBILIDADE URBANA

**LAVRAS – MG**

**2014**

**RODRIGO FERREIRA DOS SANTOS GONÇALVES**

# DESENVOLVIMENTO DE *WEB SERVICE* PARA APLICATIVO BASEADO EM LOCALIZAÇÃO GEOGRÁFICA APLICADO À MOBILIDADE URBANA

Relatório de estágio supervisionado apresentado ao Colegiado do Curso de Ciência da Computação para obtenção do título de Bacharel em "Ciência da Computação"

Orientador
Profa. Dra. Ana Paula Piovesan Melchiori

Co-Orientador
Ricardo Alexandre da Silva Shimabukuro
Victório

**LAVRAS – MG**

**2014**

**RODRIGO FERREIRA DOS SANTOS GONÇALVES**

# DESENVOLVIMENTO DE WEB SERVICE PARA APLICATIVO BASEADO EM LOCALIZAÇÃO GEOGRÁFICA APLICADO À MOBILIDADE URBANA

Trabalho de Conclusão de Curso de Graduação apresentado ao Colegiado do Curso de Bacharelado em Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 19 de novembro de 2014.

Dr. Neumar Costa Malheiros

Drª Marluce Rodrigues Pereira

Drª Ana Paula Piovesan Melchiori (Orientadora)

Ricardo Alexandre da Silva Shimabukuro Victório (Coorientador)

**LAVRAS-MG**
**Novembro/2014**

*Dedico este trabalho às minhas mães por terem me ensinado a voar:*

*à Djanira, por pular de cabeça;*

*à Margarida, pelo salto de fé;*

*e à Eva, por me lembrar do paraquedas.*

*Dedico também aos Santos, por me acolherem em sua família e aos meus amigos*

*por me ajudarem a resolver meus problemas:*

*Ao Leandro, pelos de saúde;*

*À Anna, pelos filosóficos;*

*À Daphine, pelos sociais;*

*Ao Mário, pelos políticos;*

*E ao Greg, pelos existenciais.*

*C’est en forgeant qu’on devient forgeron.*

*(Provérbio Francês)*

## RESUMO

A criação de aplicativos móveis muitas vezes impõe a necessidade de um sistema distribuído. O presente trabalho relata as experiências de estágio ocorridas em uma empresa de tecnologia durante o desenvolvimento de um desses sistemas: um *web service*.



**Palavras-Chave:** Mobilidade; Geolocalização; *Web Service*.

**SUMÁRIO**

## LISTA DE FIGURAS

# 1 ESTÁGIO

O estágio, realizado na empresa Mitah Technologies, tem por objetivo geral proporcionar formação técnico-científica, social e profissional sem a geração de vínculo empregatício entre as partes, de acordo com o art. 10 da Lei nº11.788/2008. Iniciado em 01/04/2014, teve duração total de 720 horas.

Além disso, o estágio tem como objetivo específico capacitar o estudante a atuar em todo o processo de desenvolvimento de aplicativos para a *web* e para dispositivos móveis.

A carga horária semanal do estágio supervisionado não obrigatório é de 20 horas semanais, não conflitantes com os horários das disciplinas matriculadas cujos créditos semanais são inferiores a 20.

A remuneração mensal é no valor de R$500,00.

## 1.1 Critérios de avaliação do estágio

Ao final de cada semana, é realizada uma breve reunião de acompanhamento, na qual cada membro da equipe descreve as atividades realizadas, desafios encontrados e descobertas relevantes ao projeto.

Além disso, trimestralmente a equipe de desenvolvimento e o orientador avaliam o desempenho do estagiário segundo seu comportamento, bem como a partir da avaliação da qualidade dos artefatos (documentos, diagramas, wireframes, código fonte) criados por ele.

Para a avaliação do comportamento do estagiário são usados os seguintes critérios: comunicação oral e escrita; autodidatismo; pró atividade; produtividade; trabalho em equipe; comprometimento; criatividade.

Já para a avaliação dos artefatos criados pelo estagiário são empregados os seguintes critérios: corretude; manutenabilidade e usabilidade. Para ambas as avaliações será usada a escala de 1 (muito ruim) a 5 (excelente).

Das avaliações ocorridas, a menor nota entre todos os critérios foi de 85%. E dentre os comentários, que são anônimos, se destacam:

> "Vem se mostrando um profissional pronto, logo de cara já apresentou sua opinião sobre os problemas que apareciam, não se intimidou pelo pouco tempo de empresa e nem por ser estagiário."

> "O Rodrigo é um excelente colega de trabalho, muito esforçado, responsável, auto-didata. Extremamente comunicativo, sempre nos procura para trocar ideias e tirar dúvidas, o que é bom, porque sempre estamos sabendo em que ele está trabalhando e, de certa forma, também aprendemos com ele. Espero que ele continue assim."

## 1.2 Empresa

A Mitah Technologies foi fundada em meados de 2007 pelo engenheiro de computação Ricardo Victório e pelo professor Doutor André Saúde. Ela possui uma equipe de doze colaboradores, composta por um doutor, dois mestres, além de graduados e estudantes dos cursos de Ciência da Computação e Sistemas de Informação da UFLA.

Atualmente incubada na Incubadora de Base Tecnológica da Universidade Federal de Lavras (INBATEC/UFLA) (Figura 1.1), a Mitah dispõe de um espaço exclusivo de aproximadamente 50m$^2$ (Figura 1.2), além de mais 200m$^2$ de espaço compartilhados com outras oito empresas de base tecnológica, o núcleo de inovação tecnológica da UFLA (NINTEC) e a coordenação do Parque Tecnológico de Lavras (LAVRASTEC).

No passado, a empresa atuava principalmente no desenvolvimento de produtos tecnológicos para o agronegócio e para a indústria de alimentos e na prestação de serviços associados a seus produtos. No entanto, atualmente, atua no


**Figura 1.1:** Incubadora de Base Tecnológica da UFLA

desenvolvimento de produtos e soluções de Tecnologia da Informação (TI) destinados a mercados diversos.

Em 2013, a Mitah criou o Leva Lá e ingressou no programa Startup Brasil, sendo acelerada pela aceleradora MGTI. O Leva Lá é um sistema aplicado à logística de carga, que faz a ligação direta entre embarcadores e transportadores autônomos em uma plataforma de *social commerce* e aplicativos móveis.

## 1.3 Projeto

O estagiário exerceu suas atividades, descritas no capítulo 4, em um novo projeto da empresa: o desenvolvimento de um aplicativo baseado em localização geográfica aplicado à mobilidade urbana. Chamado de Leva Eu, o aplicativo se trata de uma ferramenta de *social commerce* de transporte de pessoas.

**Figura 1.2:** Espaço exclusivo da Mitah Technologies

O Leva Eu tem por objetivo otimizar os serviços de transporte, aumentar a lucratividade do transportador, diminuir o tempo de trajeto do usuário, reduzir o uso do transporte individual e, consequentemente, ajudar a frear a tendência de caos no trânsito urbano.

## 1.4 Atuação da empresa na área de Computação

A Mitah Technologies atua no desenvolvimento de sistemas ERP (*Enterprise Resource Planning* — Sistema Integrado de Gestão Empresarial) para microempresas, e no desenvolvimento de soluções para transporte de carga e de pessoas.

Um ERP é uma plataforma de software que permite o gerenciamento e armazenamento de todas as informações de negócios de uma empresa afim de integrar os processos dos seus diversos departamentos (como finanças, contabilidade, vendas, etc).

Das soluções de transporte, “o Leva Lá é uma plataforma de negócios que conecta embarcadores a transportadores de carga em todo o país” (LEVALÁ, 2014). Já o Leva Eu é uma plataforma de negócio que conecta passageiros a prestadores de serviço (agenciadores, transportadores e intérpretes). Ambas são soluções *web*, sendo que o Leva Eu terá aplicativos para os sistemas operacionais móveis iOS e Android.

Por serem soluções *web*, o modelo cliente-servidor é usado e existe uma separação clara entre desenvolvimento *front-end* (cliente) e *back-end* (servidor). Portanto, a área de engenharia de software tem impacto forte no dia a dia da empresa. Assim como a área de sistemas distribuídos, uma vez que os compontentes do sistema (banco de dados, servidores, clientes) estão distribuídos geograficamente. Além disso, a área de interação humano-computador é fundamental no desenvolvimento *front-end*.

A empresa utiliza a metodologia DDD (*Domain-driven design*, *design* orientado a domínios). O DDD é uma abordagem para o desenvolvimento de softwares em que a implementação ocorre sobre um modelo em evolução (HAYWOOD, 2009). Ou seja, o DDD utiliza um conjunto de padrões para criar aplicações a partir do modelo, sendo esse o centro do desenvolvimento. Para isso, a empresa utiliza o Grails, um framework para desenvolvimento de aplicações *web* que comporta esse tipo de metodologia.

# 2 *WEB SERVICES*

O World Wide Web Consortium (W3C) define web service como "um sistema de software projetado para suportar interação interoperável máquina-a-máquina através de uma rede" (W3C, 2004b). Além disso, o W3C Web Service Architecture Working Group (WSAWG), que é o grupo de trabalho responsável pela arquitetura web service, estende essa definição da seguinte forma (W3C, 2004b):

> "[*web service*] tem uma interface descrita num formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o Web service de uma maneira prescrita por sua descrição usando mensagens SOAP, normalmente transmitidas usando HTTP com uma serialização XML em conjunto com outros padrões Web."

O próprio WSAWG reconhece que existem várias definições, sendo essa a que escolheram para usar no documento deles. Na Mitah, são usadas tecnologias diferentes das citadas anteriormente. Apesar disso, essa definição é um bom ponto de partida para a discussão deste trabalho, particularmente por citar XML, WSDL, HTTP e SOAP, definidos a seguir.

XML é uma linguagem de marcação para criação de documentos legíveis para humanos e máquinas. "Originalmente concebido para enfrentar os desafios da publicação eletrônica em grande escala, XML está desempenhando um papel cada vez mais importante na troca de uma ampla variedade de dados na Web e em outros lugares" (W3C, 2014). Uma alternativa ao XML é o JSON (*JavaScript Object Notation*), "um formato de troca de dados leve, baseado em texto e independente de linguagem" (IETF, 2014).

WSDL (*Web Service Description Language*) "fornece um modelo e um formato XML para descrever Web services" (W3C, 2007b). Esse arquivo XML descreve como o serviço pode ser chamado, quais parâmetros espera, e quais estruturas de dados retorna.

HTTP (*Hypertext Transfer Protocol*) "é um protocolo de aplicação para sistemas de informação de hipermídia, distribuídos e colaborativos" (ISOC, 1999). É a base de comunicação de dados para a *World Wide Web*.

SOAP (*Simple Object Access Protocol*) é "um protocolo destinado à troca de informações estruturadas em um ambiente distribuído e descentralizado" (W3C, 2007a). Além disso, utiliza XML para formatar as mensagens e protocolos da camada de aplicação (particularmente o HTTP) para a negociação e transmissão das mensagens.

Em outras palavras, um Web service na definição da W3C possui XML como formato para troca de dados, WSDL como API (*Application Programming Interface*), HTTP para transmissão de dados, e SOAP como um protocolo que padroniza tudo isso.

Na próxima seção, será apresentado um estilo arquitetural para criação de *web services* chamado REST. É importante salientar que, apesar de, por muito tempo, qualquer API que não usasse SOAP fosse comercializada como REST, o "SOAP 1.2 pode ser usado de maneira consistente com REST" (W3C, 2004a).

## 2.1 REST

REST (*Representational State Transfer*, Transferência de Estado Representativo, em português) é um estilo arquitetural que consiste de um conjunto coordenado de restrições aplicados a um sistema hipermídia distribuído (RICHARDSON; RUBY, 2007).

O termo foi cunhado por Roy Fielding em 2000, em sua tese de doutorado (RICHARDSON; RUBY, 2007). Quando um *web service* segue os princípios REST, ou seja, se conforma às suas restrições, ele é chamado de RESTful. A seguir são descritas essas restrições:

**Modelo cliente-servidor:** O modelo cliente-servidor, usado para melhorar a portabilidade, separa as responsabilidades entre as máquinas que fazem requisições e aquela que as responde. Por exemplo, o cliente não sabe nem se preocupa com a forma que os dados são armazenados no servidor, enquanto este desconhece a interface do usuário ou os estados do cliente.

***Stateless*:** *Stateless* (sem estado) é a restrição que impõe que cada requisição do cliente para o servidor precisa conter toda informação necessária para que ela seja entendida. Ou seja, o servidor não mantém estado, ou sessão, do cliente.

**Cache:** Cache, usada para melhorar a eficiência da rede, é a restrição que requer que os dados contidos numa resposta de uma requisição seja implícita ou explicitamente rotulada como cacheáveis ou não.

**Sistema em camadas:** Sistema em camadas, usado para melhorar a escalabilidade e reforçar políticas de segurança, é a restrição que previne um cliente de distinguir se está diretamente conectado ao servidor final ou a um servidor intermediário.

**Código sob demanda:** Código sob demanda, usado para melhorar a extensibilidade do sistema, permite clientes obter (através de *download*) e executar código na forma de *applets* e *scripts*. Essa é a única restrição opcional do REST.

**Interface uniforme:** Interface uniforme, usado para simplificar e desacoplar os componentes da arquitetura, permitindo que eles evoluam de forma independente. Essa é a restrição que distingue REST dos outros estilos arquiteturais, e é composta dos seguintes princípios:

**Identificação de recursos:** Qualquer informação que possa ser nomeada, pode também ser um recurso. Cada recurso tem um identificador, que

no caso de web service é um URL (*Uniform Resource Locator*, Localizador Uniforme de Recurso). Alguns recursos são estáticos, outros são dinâmicos. Por exemplo, um web service fictício sobre música chamado webmusic poderia possuir os seguintes URLs: `https://webmusic.com/albums/madonna/hard_candy` (recurso estático) e `https://webmusic.com/albums/madonna/latest` (recurso dinâmico). O primeiro representa o álbum Hard Candy da Madonna, já o segundo, o último álbum lançado pela cantora. Em 2009, os dois URLs apresentariam a mesma informação, sendo que à partir de 23 de março de 2012, o URL `https://webmusic.com/albums/madonna/latest` passaria a apresentar informações do álbum MDNA, enquanto que `https://webmusic.com/albums/madonna/hard_candy`, continuaria apresentando o álbum Hard Candy. Apesar de ambos URLs apresentarem a mesma informação em um determinado momento, eles representam recursos diferentes, e, por isso, devem possuir identificadores distintos.

**Representação:** Uma representação é uma sequência de bytes mais os metadados que descrevem esta sequência. Um cliente executa ações sobre um recurso através de representações, afim de obter seu estado atual, ou modificá-lo transferindo uma representação do estado desejado.

**Mensagens auto-descritivas:** Cada mensagem possui informações suficientes para descrever como processar a mensagem. Para um web service, isso significa adicionar informações do tipo de mídia de internet (tipo MIME).

**HATEOAS:** Através de *Hypermedia as the engine of application state* (HATEOAS), Hipermídia como o motor do estado da aplicação, clientes executam transições de estado (navegação entre recursos e manipulação dos mesmos) somente através de ações que são dinamicamente

> identificadas em hipermídia pelo servidor. Ou seja, exceto por alguns simples pontos de entrada para a aplicação, um cliente não assume que nenhuma ação está disponível para nenhum recurso além daquelas descritas em representações anteriormente recebidas do servidor.

## 2.2 *Web services* usados no projeto

Essa seção descorre sobre os *web services* usados no projeto e a função que cada um desempenha, além de relacionar algumas de suas características às definições de *web service* descritas nas últimas duas seções.

O WSAWG identifica duas classes principais de *Web services*: REST e arbitrários (W3C, 2004a). Enquanto o estilo REST usa um conjunto de operações *stateless*, *web services* arbitrários expõem um conjunto arbitrário de operações. Como visto na seção anterior, existem outras restrições além de interface uniforme a serem seguidas numa arquitetura REST. Apesar disso, todos os *web services* usados no projeto são anunciados como RESTful devido à conformidade a essa restrição, mesmo quando ela não é seguida 100%.

### 2.2.1 Flickr

O Flickr é uma aplicação web e *web service* de hospedagem de imagem e vídeo. Ele é usado no projeto não só por eliminar a necessidade de armazenamento das imagens no servidor do Leva Eu, mas também por permitir buscar fotos através de vários parâmetros. A capacidade de buscar imagens com licença de uso comercial, filtrar as que não são apropriadas para todas as idades, e delimitar uma região geográfica em que as fotos foram tiradas, são particularmente úteis para sugerir imagens que representem uma excursão no momento de sua criação.

O Flickr disponibiliza sua API em SOAP, XML-RPC (que utiliza chamadas a procedimentos remotos usando XML como formato), e REST, sendo essa úl-

tima a usada no “Leva Eu”. É interessante notar que o Flickr documenta sua API[1] de forma consistente para os três estilos. Isso causa alguns efeitos estranhos sobre a API REST, como a inclusão de um método HTTP no URL para identificar a ação sobre o recurso, enquanto que na verdade é usado outro. Por exemplo, para excluir um comentário de uma foto, deve-se enviar uma requisição contendo o identificador do comentário usando o método POST para o URL `https://www.flickr.com/services/api/flickr.photos.comments.deleteComment`. Numa implementação purista do REST, deveria ser feito uma requisição usando o método DELETE para um URL no formato `https://www.flickr.com/services/api/comments/{comment_id}` ou para `https://www.flickr.com/services/api/photos/{photo_id}/commments/{comment_id}`, onde `{comment_id}` é o identificador do comentário e `{photo_id}` o identificador da foto.

Na documentação da API, são disponibilizadas várias bibliotecas escritas por desenvolvedores em várias linguagens, que têm por objetivo abstrair a comunicação através da rede, ou seja, as trocas de mensagens entre clientes e servidores. No “Leva Eu”, foi usada uma dessas bibliotecas para a linguagem Java, chamada Flickr4Java[1].

### 2.2.2 OpenStreetMap

O OpenStreetMap é um projeto colaborativo que tem por objetivo criar um mapa livre, no sentido de software livre, e editável do mundo. Ele é usado no projeto para encontrar as coordenadas geográficas de um lugar (dado parte de seu endereço), determinar rotas e calcular distâncias entre esses lugares. A integração do OpenStreetMap foi feita por um outro colaborador da empresa, não pelo autor desse relatório, portanto a relação de suas características com as definições de *web service* não são descritas aqui.

[1] `https://www.flickr.com/services/api/`
[1] `https://github.com/callmeal/Flickr4Java`

### 2.2.3 Paypal

O Paypal é um sistema que permite transferência de dinheiro e pagamento eletrônico. Ele é usado no projeto como um *gateway* de pagamento. Sua escolha se deu principalmente por sua popularidade internacional, que por sua vez foi importante já que o "Leva Eu" foi lançado durante a Copa do Mundo FIFA de 2014.

O Paypal disponibiliza sua API[2] apenas em formato REST e todas as requisições e respostas são formatadas em JSON. Além isso, sua implementação segue atentamente as restrições desse estilo arquitetural, inclusive quanto a HATEOAS. Cada resposta inclui um conjunto de links HATEOAS contendo os seguintes atributos:

- **href**: URL do link HATEOAS
- **rel**: uma descrição da relação entre o recurso atual e o link HATEOAS.
- **method**: o método HTTP que deve ser usado na requisição.

Na documentação da API, também são disponibilizadas várias bibliotecas, dentre elas uma para a linguagem Java. Apesar disso, foi decidido não usar a biblioteca pela forma em que o *token* de acesso era obtido e usado. É necessário fazer uma requisição ao *web service* para obter um *token* de acesso que é usado para fazer requisições aos outros recursos e é válido por um tempo determinado pelo PayPal. Esse tempo está contido na resposta da requisição de obtenção do *token*. No entanto, na biblioteca, em vez de reutilizar o *token* pelo tempo que ele é válido, um novo é gerado para cada requisição feita ao *web service*.

### 2.2.4 Iugu

O Iugu é um sistema brasileiro similar ao Paypal, que permite pagamento através de boleto bancário e cartão de crédito, incluindo parcelamento de fatura. Apesar do

[2] https://developer.paypal.com/webapps/developer/docs/api/

Paypal possuir taxas menores nos pagamentos via cartão de crédito, nem todas as funcionalidades de sua API RESTful estão disponíveis no Brasil. Por exemplo, o pagamento direto via cartão de crédito, que permite ao sistema ("Leva Eu") coletar os dados de cartão de crédito (sem os armazenar, obviamente) e os enviar para o *gateway* de pagamento em vez de redirecionar o usuário para o website do *gateway* (onde os dados seriam coletados), melhora a experiência do usuário na plataforma móvel. Essa funcionalidade não está disponível no Paypal para o Brasil, mas está no Iugu.

O Iugu disponibiliza sua API[1] apenas em formato REST. Além disso, são disponibilizadas bibliotecas para as linguagens Ruby e PHP. Como o "Leva Eu" não está sendo desenvolvido em nenhuma dessas duas linguagens, nenhuma dessas bibliotecas foi usada. Em vez disso, a comunicação foi feita usando a biblioteca HTTP Builder[2].

### 2.2.5 Arquitetura do sistema de pagamento do Leva Eu

A interação com os meios de pagamento foi implementada em camadas, como mostra a figura 2.1.

No nível mais alto, e o único visível para os clientes da biblioteca (no caso, o controlador), está o Sistema de pagamento. Essa camada serve para abstrair os *gateways* de pagamento, além de persistir as informações no banco de dados. Na camada abaixo estão o Serviço Iugu e o Serviço PayPal, que servem para abstrair a comunicação com os *web services* dos *gateways* de pagamento. Essa camada é responsável por fazer autenticação, executar as requisições HTTP e formatar as respostas para a camada de cima.

Essa arquitetura em camadas visa, além da separação de responsabilidades, o encapsulamento do sistema de pagamentos, tornando-o transparente para o cliente. Uma das vantagens de tal transparência no "Leva Eu" é a possibilidade de

[1] `http://iugu.com/referencias/api`
[2] `http://groovy.codehaus.org/modules/http-builder/home.html`

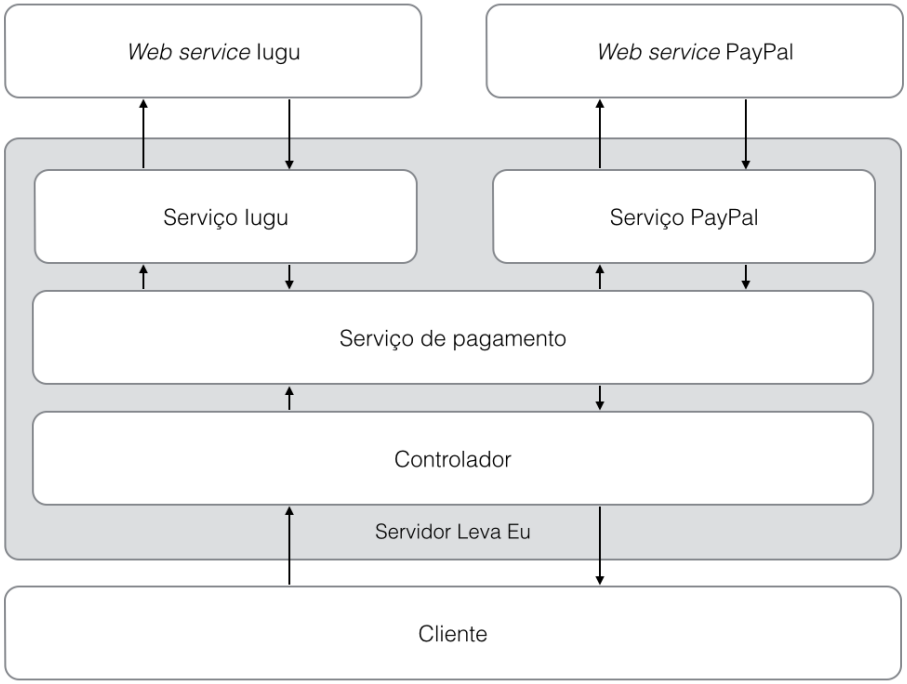


**Figura 2.1:** Arquitetura dos sistema de pagamento do Leva Eu

adicionar ou substituir *gateways* de pagamento sem que o cliente tenha que modificar o seu código. Desta forma, um *gateway* pode ser usado no lugar do outro caso as taxas sejam mais favoráveis, por exemplo.

# 3 PROCESSOS TÉCNICOS

Esse capítulo descreve o processo de desenvolvimento de *web services* executado na empresa e apresenta as tecnologias e ferramentas utilizadas nesse processo.

## 3.1 Processo de desenvolvimento de *web services*

As figuras a seguir mostram o fluxograma do processo de desenvolvimento de *web services* na Mitah. Nela, foram separadas as atividades que são executadas pelas pessoas com os seguintes papéis: líder da equipe, desenvolvedor e desenvolvedor de teste. Além disso, no fluxograma aparece o papel do analista de requisitos, porém como a participação dele nesse processo é somente de consulta, suas atividades não foram descritas aqui.

## 3.2 Tecnologias

Esta seção descreve as principais tecnologias usadas no processo de desenvolvimento de *web services* da empresa.

**Grails:** O Grails é um *framework* de desenvolvimento rápido que roda sobre uma máquina virtual Java (JVM), foi inspirado no *framework* Ruby on Rails e usa uma linguagem dinâmica, o Groovy (JUDD; NUSAIRAT; SHINGLER, 2008). Ele utiliza o modelo de desenvolvimento de software conhecido como convenção sobre configuração (ou programação por convenção) que visa diminuir a quantidade de decisões tomadas pelo desenvolvedor.

Além disso o Grails possui as seguintes características e componentes (GOPIVOTAL, 2014):

- Um *framework web* que segue o modelo de arquitetura MVC (*Model-view-controller* — modelo-visão-controlador).

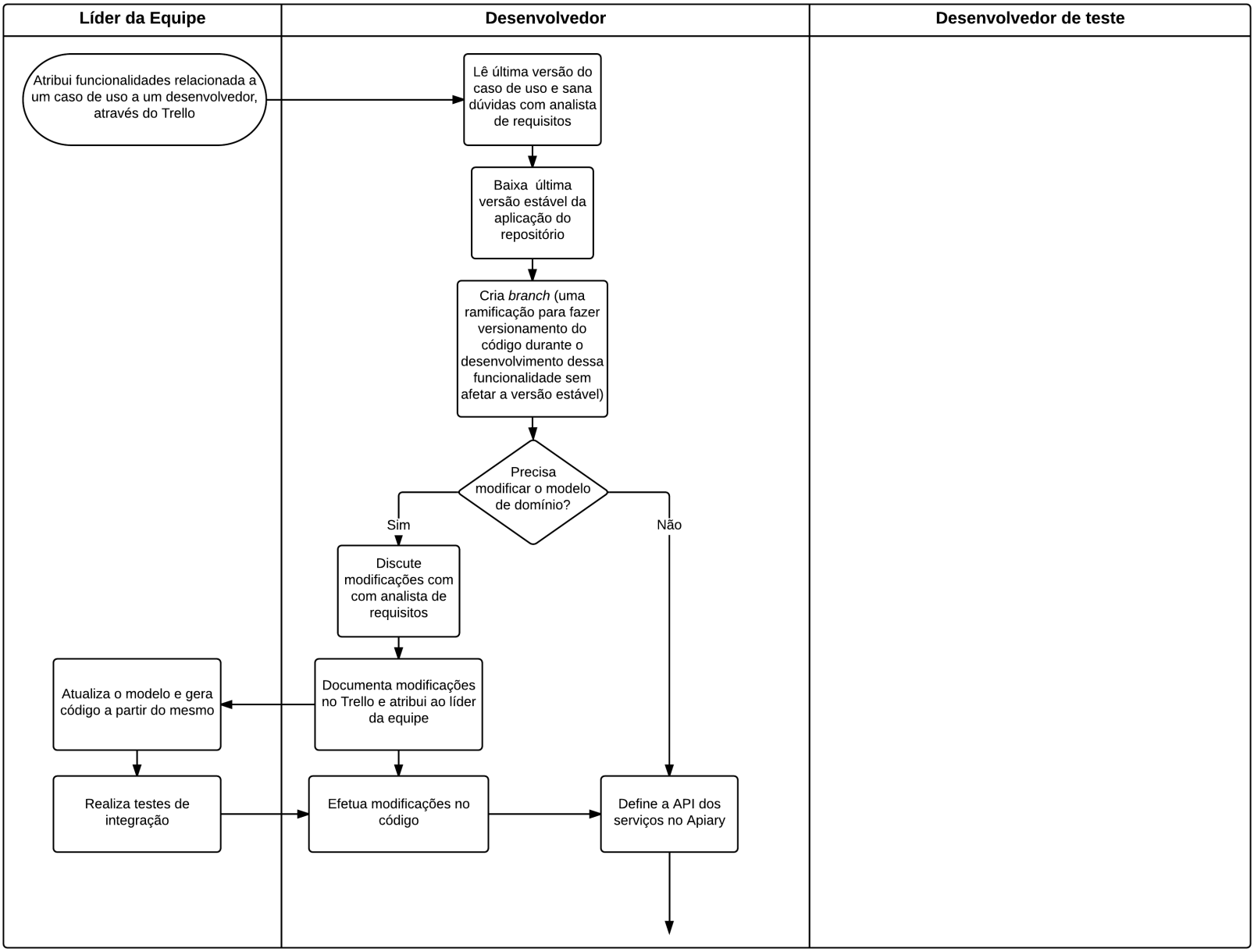


**Figura 3.1:** Processo de desenvolvimento de *web services* - parte 1

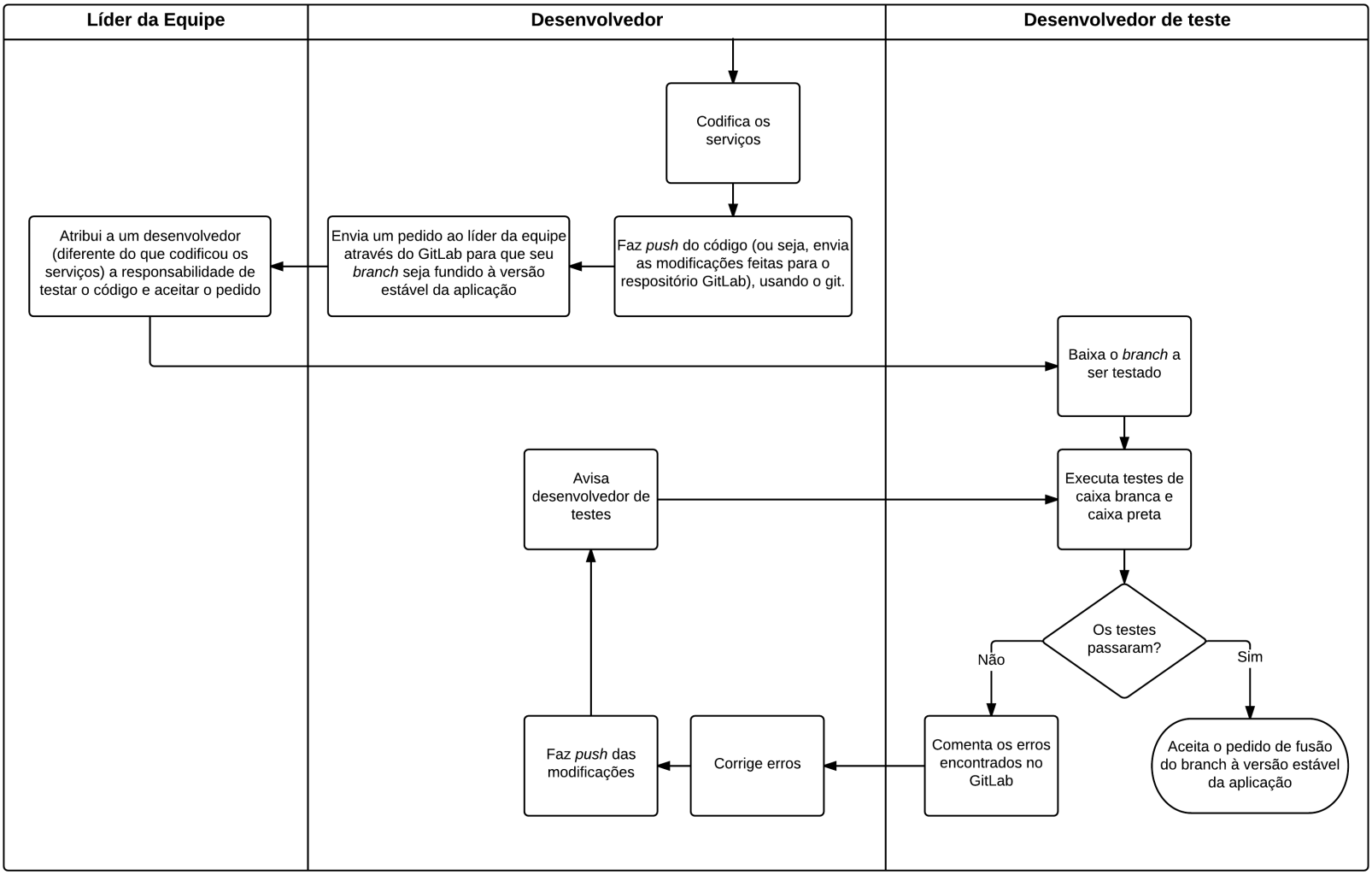


**Figura 3.2:** Processo de desenvolvimento de *web services* - parte 2

- GSP (Groovy Server Pages), uma tecnologia que ajuda os desenvolvedores a criarem páginas *web* dinâmicas (a visão do modelo MVC).
- Um *web container*, ou seja, um servidor *web* embutido.
- GORM (Grails Object Relational Mapping), uma técnica que abstrai a camada com o banco de dados.
- Um banco de dados em memória principal SQL.
- Suporte a internacionalização (i18n).
- Um sistema customizável de construção de software.
- Suporte embutido de teste.
- Um sistema de geração de documentação.

**Groovy:** Groovy é uma linguagem de programação dinâmica que pode ser interpretada ou compilada e foi desenvolvida para a plataforma Java (CHAMPEAU, 2014). Quando compilada, ela gera Java Virtual Machine (JVM) *bytecode*. Além disso, todas as classes e bibliotecas Java existentes podem ser usadas com o Groovy de forma transparente.

Apesar de toda essa integração com Java, Groovy foi influenciada por Ruby, Python, Perl e Smalltalk, incorporando várias funcionalidades não presentes em Java, como tipagem estática e dinâmica (com o uso da palavra chave def), *closures*, sintaxe nativa para listas e vetor associativos (maps), entre outros.

Ademais, Groovy possui propriedades (também conhecidas como Groovy-Beans) cujos métodos acessores e mutadores são gerados implicitamente. Groovy também oferece metaprogramação e recursos de programação funcional como *closures*, aplicação parcial de funções, avaliação preguiçosa, *reduce/fold*, estruturas infinitas e imutabilidade.

## 3.3 Ferramentas

Esta seção descreve as principais ferramentas usadas no processo de desenvolvimento de *web services* da empresa.

**Git:** O Git é um sistema de controle de versão distribuído de código aberto. Sua função é controlar as modificações dos arquivos, criando versões diferentes para cada modificação, possibilitando ao usuário voltar a uma versão anterior de um arquivo, caso necessário (CHACON, 2009).

**GitLab:** O GitLab é um sistema de código aberto que oferece gerenciamento de repositórios git, revisão de código, acompanhamento de problemas e criação de documentações Wiki. Além disso, vários usuários de um repositório podem fazer modificações em suas máquinas locais simultaneamente e, ao enviar essas modificações para o GitLab, resolver os conflitos, quando ocorrerem.

**Apiary:** O Apiary é uma aplicação online para planejar e estruturar a API de *web services*, ou seja, é possível definir, usando um editor online, os URLs dos serviços web, seus parâmetros e as respostas esperadas.

**Trello:** O Trello é uma aplicação online gratuita para gerenciamento de projetos de forma colaborativa. Nele, é possível criar tarefas, organizá-las em grupos e atribuí-las a usuários.

**Google Drive:** O Google Drive é uma aplicação online usada no projeto para armazenar e sincronizar arquivos. Em particular, os casos de uso.

# 4 ATIVIDADES

O “Leva Eu” é um projeto muito maior e complexo do que foi descrito neste documento. Tudo que foi mostrado até então está ligado de alguma forma às atividades desempenhadas pelo estagiário. A seguir, a paticipação do estagiário nessas atividades é explicada.

**Estudo de tecnologias:** Um tempo considerável, principalmente no início, foi dedicado ao estudo das tecnologias e ferramentas usadas no projeto. Além da compreensão do conteúdo teórico, a prática das tecnologias foi muito importante para o aprendizado das mesmas.

**Integração com *web service* Flickr:** as atividades de integração com o Flickr incluem: o estudo do serviço e sua API, o estudo da biblioteca Flickr4Java e a criação de um serviço dentro da aplicação “Leva Eu” que abstrai a comunicação com o *web service* afim de buscar fotos, assim como fazer *upload* e remover fotos.

**Integração com *web service* PayPal:** para fazer integração com o PayPal, o estagiário teve de estudar muito bem o serviço, as implicações financeiras e jurídicas que envolvem o serviço, além de políticas, como a de reembolso. Ademais, foi necessário entender a API e aprender a utilizar a biblioteca HTTP Builder para fazer a comunicação com o serviço. Também foi criado um serviço que abstrai essa comunicação e as funcionalidades, ou seja, foi criado uma biblioteca para criar pagamentos, liquidar parte ou todo o pagamento, além de fazer reembolso parcial ou integral.

**Integração com *web service* Iugu:** assim como com o PayPal, para fazer a integração com o Iugu, o estagiário teve de estudar as particularidades desse *gateway* de pagamento, além de seus serviços e sua API. Também foi criado um serviço para abstrair a comunicação e as suas funcionalidades, ou seja,

uma biblioteca que salva de forma segura os dados de cartão de crédito do usuário nos servidores do Iugu, realiza pagamentos através desses cartões, além de fazer reembolso dos pagamentos e criar boletos bancários.

**Sistema de Pagamento:** afim de modularizar as funcionalidades relacionadas a pagamento e abstrair com qual *gateway* (PayPal ou Iugu), o estagiário criou o sistemas em camadas explicado na seção 2.2.5.

**Desenvolvimento do *web service* "Leva Eu":** Além da integração com *web services* externos, o estagiário também fez parte do desenvolvimento interno. No processo descrito na seção 3.1, o estagiário atuou como desenvolvedor, desenvolvedor de teste, e às vezes como analista devido ao conhecimento adquirido no estudo dos *web service* Flickr, PayPal e Iugu.

# 5 CONTRIBUIÇÕES E SUGESTÕES

Esta capítulo descreve as contribuições da empresa para o estagiário e vice-versa. Além disso, são feitas algumas sugestões para trabalhos futuros no projeto.

## 5.1 Contribuições

O estagiário pretende seguir carreira no desenvolvimento de aplicativos. A empresa contribuiu de forma imensurável para esse objetivo ao expô-lo ao processo de desenvolvimento de *web service*, às decisões de projeto concernentes a esse desenvolvimento, e a um ambiente de trabalho saudável composto de uma equipe brilhante.

Uma das contribuições do estagiário para a empresa que merece ser mencionadas devido ao impacto sobre o projeto, foi quanto ao uso correto dos verbos HTTP na construção da API do *web service*.

Muitas vezes, ao buscar um conjunto de recursos, vários parâmetros são necessários para filtrar o que será retornado. Dessa forma, parece natural que uma requisição com o verbo POST seja feita, passando os parâmetros no corpo da requisição. O problema com tal abordagem, na teoria, é a de que requisições que não modificam o estado do recurso devem ser feitas usando o verbo GET. Já na prática, o problema é a falsa impressão de que o verbo GET não é necessário, uma vez que os efeitos desejados estão sendo obtidos apenas com o verbo POST. A equipe de desenvolvimento do "Leva Eu" caiu nessa armadilha.

O estagiário vinha estudando a teoria por trás de *web services* RESTful, e ao perceber a má prática, alertou a equipe. Como mudanças consomem tempo, o estagiário teve de argumentar o impacto prático sobre o projeto, afim de justificar tais mudanças. Ele conseguiu convencer a equipe ao lembrar a todos que não seria possível gerar *links* para os recursos a menos que o verbo GET fosse usado, ou

seja, sem a mudança, não seria possível que os usuários salvassem uma página nos favoritos de seu navegador ou compartilhassem o *link* com outras pessoas.

## 5.2 Sugestões

Enquanto este trabalho é escrito, o projeto "Leva eu" está em andamento. Além disso, por se tratar de um projeto grande, as oportunidades de vários mercados serão exploradas num momento posterior à saída do estagiário da empresa. Desta forma, esta seção sugere algumas orientações para trabalhos futuros.

Uma dessas oportunidades se refere à abertura da API para que outros sistemas possam integrar a contratação dos serviços de transporte de pessoas ao seus processos. Por exemplo, um sistema de reservas de hospedagem poderia usar a API do "Leva Eu" para dar a opção de translado entre o aeroporto e o hotel.

Como o desenvolvimento do "Leva Eu" é orientado a domínio (*Domain Driven Development*), e como por padrão os URLs são gerados a partir dos *controllers* desses domínios, sua API pode ser um pouco confusa de entender. Por exemplo, `www.levaeu.net/api/city` lista cidades e `www.levaeu.net/api/cityCustom/search` busca por cidades. Embora, os dois URLs identificam recursos relacionados a cidades, o segundo é composto por `cityCustom` enquanto o primeiro simplesmente por `city`. Isso pode levar o desenvolvedor da aplicação cliente a cometer erros desnecessários. Ademais, o termo `Custom` em `cityCustom` diz respeito à estrutura interna do *web service*, e não possui relevância alguma para o cliente da aplicação.

Hoje, essa API é fechada para uso exclusivo das plataformas do próprio sistema. Portanto, essa peculiaridade é bem conhecida pelos desenvolvedores da aplicação cliente. Porém, ao abrir a API para uso externo, é recomendável uma maior atenção à uniformidade da interface afim de simplificar e desacoplar a arquitetura.

Esta mudança pode ser feita através do uso de URL Mappings (mapeamentos de URL) do Grails. Como já mencionado, por padrão os URLs são mapeados para métodos nos *controllers* de mesmo nome. Por exemplo, `www.levaeu.net/api/city` é mapeado para o método `index` do *controller* `city` e `www.levaeu.net/api/cityCustom/search` para o método `search` do *controller* `cityCustom`. Mas, novos mapeamentos podem ser criados. Seguindo as boas práticas do REST, os recursos citados também poderiam ser acessados pelos URLs `www.levaeu.net/api/cities` e `www.levaeu.net/api/cities/search`.

Concluindo, a API pode ser exposta de maneira uniforme ao criar novos mapeamentos e nenhum cliente precisará ser modificado ao manter os mapeamentos antigos.

# 6 CONCLUSÃO

O objetivo deste documento foi relatar as experiências do estágio supervisionado ocorrido na empresa Mitah Technologies. Mais especificamente, o trabalho realizado no desenvolvimento de *web service* do aplicativo baseado em localização geográfica aplicado à mobilidade urbana, o projeto "Leva Eu".

Conclui-se dessas experiências que a teoria é fundamental para basear, guiar e validar o desenvolvimento de aplicativos. Porém, é com a prática que o conhecimento é consolidado e internalizado de forma a ser usado de forma instintiva, qualidade essencial para atuar de forma efetiva no mercado de trabalho atual.

# REFERÊNCIAS BIBLIOGRÁFICAS

CHACON, S. *Pro Git*. 1st. ed. Berkely, CA, USA: Apress, 2009. ISBN 1430218339, 9781430218333.

CHAMPEAU, C. *Groovy...* 2014. Disponível em: http://groovy.codehaus.org.

GOPIVOTAL. *Learn Grails in 5 easy steps*. 2014. Disponível em: https://grails.org/learn.

HAYWOOD, D. *An Introduction to Domain Driven Design*. 2009. Disponível em: http://www.methodsandtools.com/archive/archive.php?id=97.

IETF. *The JavaScript Object Notation (JSON) Data Interchange Format*. 2014. Disponível em: http://tools.ietf.org/html/rfc7159.

ISOC. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. Disponível em: https://tools.ietf.org/html/rfc2616.

JUDD, C. M.; NUSAIRAT, J. F.; SHINGLER, J. *Beginning Groovy and Grails: From Novice to Professional*. 1. ed. Berkely, CA, USA: Apress, 2008. ISBN 1430210451, 9781430210450.

LEVALÁ. *Leva Lá*. 2014. Disponível em: http://levala.com.br/.

RICHARDSON, L.; RUBY, S. *Restful Web Services*. First. [S.l.]: O'Reilly, 2007. ISBN 9780596529260.

W3C. *Web Services Architecture*. 2004. Disponível em: http://www.w3.org/TR/ws-arch/#relwwwrest.

W3C. *Web Services Glossary*. 2004. Disponível em: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. Disponível em: http://www.w3.org/TR/soap12-part1/#intro.

W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2007. Disponível em: http://www.w3.org/TR/wsdl20/#intro.

W3C. *Extensible Markup Language (XML)*. 2014. Disponível em: http://www.w3.org/XML/.