CENTRO UNIVERSITÁRIO DO TRIÂNGULO
PÓS-GRADUAÇÃO EM DESENVOLVIMENTO JAVA

Desenvolvimento de Sistema de Gerenciamento de Vendas utilizando a Plataforma Android e Java EE Consumindo um WebService RESTful

Bruno Stort Silva

Gladston Lázaro Parreira

Lucas Freitas Terra Silva

Rodrigo Gonçalves da Silva

Uberlândia, Julho de 2014

CENTRO UNIVERSITÁRIO DO TRIÂNGULO
PÓS-GRADUAÇÃO EM DESENVOLVIMENTO JAVA

Desenvolvimento de Sistema de Gerenciamento de Vendas utilizando a Plataforma Android e Java EE Consumindo um WebService RESTful

Bruno Stort Silva

Gladston Lázaro Parreira

Lucas Freitas Terra Silva

Rodrigo Gonçalves da Silva

Trabalho de conclusão de curso apresentado ao Curso de Pós-Graduação *lato sensu* em Desenvolvimento Java do Centro Universitário do Triângulo - Unitri, como requisito básico à obtenção do título de Especialista em Desenvolvimento Java, sob a orientação dos professores Dr. Marcos Alberto Lopes da Silva e MSc. Sônia Aparecida Santana.

Uberlândia, Julho de 2014.

Desenvolvimento de Sistema de Gerenciamento de Vendas utilizando a Plataforma Android e Java EE Consumindo um WebService RESTful

Bruno Stort Silva

Gladston Lázaro Parreira

Lucas Freitas Terra Silva

Rodrigo Gonçalves da Silva

Trabalho de conclusão de curso apresentado ao Curso de Pós-Graduação *lato sensu* em Desenvolvimento Java do Centro Universitário do Triângulo - Unitri, como requisito básico à obtenção do título de Especialista em Desenvolvimento Java.

Sônia Aparecida Santana, Msc.
(Orientador)

Marco Alberto Lopes da Silva, Dr.
(Orientador)

Marcos Alberto Lopes da Silva, Dr.
(Coordenador de Curso)

Uberlândia, Julho de 2014.

Agradecemos primeiramente a Deus que nos concedeu a graça deste curso, a todos os professores que foram nossos mestres e guias, a todos os colegas, amigos e familiares que nos ajudaram.

RESUMO

Este trabalho apresenta um estudo sobre tecnologias Java para o desenvolvimento de aplicações na plataforma *Google Android* e Java EE, ambas consumindo um *WebService* construído no estilo arquitetural REST (*RESTful*). O detalhamento dos principais conceitos envolvidos é realizado de forma a se demonstrar o estado atual da arte e destacar como algumas tecnologias vêm evoluindo rapidamente. Atualmente, tanto aplicações *web* quanto aplicativos móveis estão bem difundidos no mercado, portanto é importante sempre conhecer plataformas e *frameworks* que colaborem para um desenvolvimento produtivo e de qualidade. A fim de demonstrar o uso das tecnologias descritas neste trabalho, selecionou-se uma microempresa, para a qual foi desenvolvido um sistema que atendesse suas necessidades reais e apresentasse na prática a integração entre as aplicações. Tendo como resultado do projeto não somente as aplicações, mas também o aprendizado proveniente de pesquisas, experiências e utilização das tecnologias.

SUMÁRIO

1. INTRODUÇÃO

Ao decorrer dos anos a tecnologia vem se tornando cada vez mais presente e essencial para o ser humano, tanto no contexto pessoal, quanto no meio corporativo.

Atualmente, os dispositivos móveis se tornaram parte do dia-a-dia devido à facilidade de uso e a capacidade de oferecer entretenimento, comunicação, conectividade e utilidade ao alcance de um toque.

Diante deste cenário é importante que as organizações avaliem se a mobilidade pode colaborar para que seus processos sejam mais ágeis, visando resultados com projeção de crescimento.

Empresas, obviamente, buscam sempre mais lucro e os dispositivos móveis ocupam um importante espaço em um mundo onde a palavra "mobilidade" está cada vez mais conhecida [LECHETA, 2010].

Este trabalho apresenta a construção de um sistema para uma microempresa de doces artesanais, a qual se encontra com dificuldades em controlar seu processo de vendas. Tal sistema é composto por uma aplicação para dispositivos móveis, desenvolvida na plataforma *Google Android*, com funcionalidades que permitem ao usuário vendedor realizar cadastros e lançamento de vendas, uma aplicação *Java Web*, a qual possibilita que o usuário administrador e cliente visualizem relatórios, e por fim um *WebService RESTful*, desenvolvido igualmente na linguagem Java, o qual será consumido pelas aplicações móvel e web.

Nas primeiras seções são apresentados assuntos como a escolha do tema com os seus respectivos objetivos e justificativas, assim como também a revisão da literatura acerca dos conceitos e ferramentas utilizadas no desenvolvimento. Na

metodologia são descritos os pontos importantes na construção do software, e ao final, são relatadas as conclusões inerentes ao trabalho.

2. A ESCOLHA DO TEMA

O tema deste trabalho foi escolhido considerando o conteúdo estudado nas disciplinas cursadas e conhecimentos adquiridos ao decorrer do curso de Pós-Graduação em Desenvolvimento Java, assim como também foi baseado na existência de uma circunstância real, ou seja, a necessidade de se resolver um problema de ordem administrativa de uma microempresa, da área de alimentação, mais precisamente, seu gerenciamento de vendas.

Como o tema e as práticas envolvidas são de interesse de todos da equipe, o tema se torna relevante e até mesmo indicado, afinal também será possível contar com o auxílio de profissionais do curso, capacitados nos assuntos abordados, permitindo dessa forma, que novas experiências sejam adquiras e conhecimentos agregados.

2.1. Delimitação do Tema

Os conceitos correspondentes às tecnologias utilizadas serão apresentados e detalhados na revisão da literatura, provenientes de fontes de pesquisas realizadas, de forma a auxiliar a compreensão do trabalho.

Na parte prática, o sistema a ser construído contemplará um aplicativo para dispositivos móveis e outro para *web*. Sendo que, o aplicativo móvel será desenvolvido na plataforma *Google Android* e possuirá apenas os módulos de cadastros de pessoas, usuários, clientes, produtos e lançamento de vendas. O aplicativo *web* possuirá o módulo de relatório de compras realizadas por período, com seus respectivos créditos e débitos, o qual poderá ser acessado por clientes, o relatório de valores faturados por período e relatório de valores recebidos por período, os quais serão acessados pelo administrador do sistema.

Além disso, para atender aos dois aplicativos e unificar o sistema, ambas as aplicações consumirão um *WebService RESTful*.

2.2. Problematização do Tema

A empresa, objeto deste estudo, possui sede em Uberlândia-MG e, conforme mencionado, atua no ramo da alimentação, mais precisamente na fabricação e vendas de doces artesanais.

No cenário em que se encontra, notou-se a necessidade de controlar e gerenciar melhor o seu processo de vendas. Atualmente isto é realizado por meio de planilhas e blocos de anotações em papel. Apesar de ser simples e barato, este controle possui algumas desvantagens, como o desmembramento de informação em várias planilhas que não tem ligações entre si, obrigando o responsável pelas vendas a digitar a mesma informação mais de uma vez, resultando em dados redundantes e podendo gerar informações inconsistentes. Além disso, para executar um conjunto ordenado de operações, planilhas possuem recursos limitados, geralmente macros.

Há, portanto a necessidade de um software específico que atenda a estas exigências e que, permita centralizar as informações, funcionando principalmente em dispositivos móveis, pois os vendedores da empresa também atuam externamente.

Embora existam soluções no mercado, elas possuem alto custo e/ou são complexas em vista ao tamanho da empresa e não dispõem de funcionalidades que atendam as necessidades específicas da mesma.

2.3. Objetivos

O objetivo geral do projeto consiste em construir um software composto por uma aplicação móvel que permita realizar cadastros e lançamento de vendas, e também uma aplicação web onde seja possível visualizar relatórios das vendas, pagamentos e débitos pendentes de clientes.

Os objetivos específicos são citados a seguir:

- Analisar a regra de negócios da empresa escolhida;
- Construir um *WebService* baseado no conceito *REST*, para transmitir e obter dados. Sendo este, desenvolvido na linguagem Java, utilizando o *framework Jersey* e a biblioteca *Gson* da *Google*. Este *WebService* será executado sob o servidor Apache *TomCat* acessando banco de dados relacional *MySQL*, tendo como finalidade centralizar o consumo de dados evitando duplicação de código;
- Construir uma aplicação para dispositivos móveis na plataforma *Google Android*, utilizando a linguagem Java e biblioteca *Gson* para a conversão de dados no formato *JSON* recebidos na comunicação com o *WebService*. Esta aplicação deve possibilitar ao vendedor realizar a listagem, cadastro e edição de pessoas, usuários, clientes, produtos e efetuar o lançamento de vendas;
- Desenvolver uma aplicação *web* na plataforma Java EE com JSF 2.0, usando as bibliotecas *PrimeFaces* e *Gson*, onde o usuário administrador autenticado visualize relatórios de faturamento e recebimento por período, e um usuário cliente autenticado visualize as compras realizadas por período com seus respectivos créditos ou débitos.

2.4. Justificativas

Analisando a necessidade de implantação e a solução proposta, temos que:

1 – Quanto a sua importância – Sua implantação é importante tendo em vista que no cenário atual a empresa trabalha com processos manuais no lançamento e organização de suas vendas.

2 – Quanto à oportunidade – Devido a alta procura pelos seus produtos a empresa não vem conseguindo gerenciar da melhor maneira os processos, ocasionando assim, em desorganização e até mesmo em alguns casos em perda de vendas.

3 – Quanto à viabilidade – Considerando a necessidade de maior precisão dos gastos e principalmente dos lucros para futuros investimentos e

visando definir melhor os prazos para os clientes, torna-se viável o desenvolvimento e a implantação do projeto.

Existem soluções no mercado que ocasionalmente poderiam atender à demanda da empresa. Na tabela 1 é possível observar a listagem de alguns sistemas, realizando um comparativo entre seus custos e informações:

Tabela 1 – Comparativo de aplicações de gerenciamento de vendas disponíveis no mercado.

Produto	Site / Download	Custo Inicial	Custo Mensal	Observações
Gerenciador Eficaz	www.gerenciadoreficaz.com.br	R$399,00	R$0,00	Não possui módulo para dispositivos móveis.
Controle de Vendas	https://play.google.com/store/apps/details?id=br.thiagopacheco.vendas	R$0,00	R$0,00	Não possui módulo administrador e cliente.
Venda Mais	https://play.google.com/store/apps/details?id=br.vendas	R$0,00	R$0,00	Não possui módulo administrador e cliente.
AppVenda$	https://play.google.com/store/apps/details?id=air.AppVendas	R$0,00	R$0,00	Não possui módulo administrador e cliente.
Controle de Vendas – Free	https://play.google.com/store/apps/details?id=com.jlheidemann.controlevendas.free	R$0,00	R$0,00	Não possui módulo administrador e cliente.
Tuvendas	www.tuvendas.com	R$0,00	R$2,50	Interface web e armazenamento dos dados em nuvem, porém não possui módulo específico para dispositivos móveis.
Controle Vendas	www.cenize.com/Novo_Loje	R$838,92 em 12X	R$0,00	Interface web, mas não possui módulo específico para dispositivos móveis.

Diante das aplicações pesquisadas e demonstradas na tabela, foi possível concluir que:

- As soluções gratuitas não possuem módulo *web* para o administrador e cliente;

- As demais soluções tem interface web, porém não tem módulo específico para dispositivos móveis.

Portanto, nenhuma delas atende a todos os requisitos esperados. Além disso, em caso de necessidade, pode não ser possível realizar uma eventual modificação ou implicaria em custo adicional. Usando a solução proposta neste trabalho, todos os requisitos serão atendidos e ainda, como um dos desenvolvedores é também proprietário da empresa, o mesmo poderá realizar melhorias necessárias no futuro.

3. REVISÃO DA LITERATURA

Tanto para o desenvolvimento do sistema, quanto para a compreensão do trabalho como um todo, é importante conhecer as tecnologias utilizadas, permitindo dessa forma, relacionar a parte textual com a prática.

Para isto, foram realizadas pesquisas com o intuito de descrever as tecnologias abordadas e utilizadas no projeto, assim como também expor o cenário atual em que se encontram.

3.1. *Google Android*

Em 05 de novembro de 2007, a OHA (*Open Handset Alliance*) anunciou a plataforma *Android*. A *Open Handset Alliance* é um grupo formado atualmente por 84 empresas de tecnologia, liderado pela *Google*, e que se uniram para acelerar a inovação em dispositivos móveis e oferecer um solução mais barata, melhor e com mais rica experiência ao usuário [RABELLO, 2014].

O *Android* é uma poderosa, ousada e flexível plataforma de desenvolvimento para aplicativos móveis e ao mesmo tempo um sistema operacional baseado no Linux. [LECHETA, 2010].

Para desenvolver aplicações *Android* a linguagem utilizada é o Java, assim como todos os seus recursos, no entanto, não existe uma máquina virtual Java (JVM), mas uma específica chamada *Dalvik*. Sendo assim, depois de compilado o *bytecode* (*.class*) é convertido para o formato .dex (*Dalvik Executable*) que representa a aplicação *Android* compilada [PEREIRA e SILVA, 2009].

No *Android*, uma plataforma é referente a uma nova versão do sistema operacional, assim pode-se dizer que existem diversas plataformas diferentes do *Android*, sendo que, cada uma tem um código identificador chamado *API Level* [LECHETA, 2010]. Na tabela 2 é possível observar o histórico das versões:

Tabela 2 – Versões do *Android*. Adaptada de [PAULA, 2013].

Versão	Codinome	API Level
1.0	-	1
1.1	-	2
1.5	Cupcake	3
1.6	Donut	4
2.1	Eclair	5,6 e 7
2.2	Froyo	8
2.3.x	Gingerbread	9 e 10
3.x	Honeycomb	11,12 e 13
4.0.x	Ice Cream Sandwich	14 e 15
4.1, 4.2, 4.3	Jelly Bean	16,17 e 18
4.4	Kit Kat	19

Novas plataformas são lançadas periodicamente e desde suas primeiras versões, a cada atualização são realizadas novas melhorias no sistema. A seguir na figura 1, percebe-se a evolução dos dispositivos e do sistema, comparando o HTC G1, o primeiro *smartphone* a receber o *Android*, com o LG Nexus 5, que possui a última versão estável disponível até o momento.

Figura 1 – À esquerda o primeiro smartphone com *Android* 1.1, o HTC G1 e à direita o LG NEXUS *5 com Android 4.4 – KitKat.* Adaptada de [GSMARENA, 2014].

Atualmente a plataforma *Android* é a forma mais utilizada para a criação de clientes interativos. A instalação e distribuição do aplicativo é mais simples e menos onerosa que seu concorrente direto, o iOS da *Apple* e, ao contrário deste, pode-se utilizar outros canais de venda, além da *Google Play*, como a Verizon *apps* da Verizon e a *appstore* da Amazon.com [MEDNIEKS et all, 2012].

Em 25 de Junho de 2014 na conferência *Google* I/O, foi anunciada a chegada da mais nova versão do sistema operacional *Android*, a qual foi intitulada inicialmente pelo termo provisório "*Android* L". Até a finalização deste trabalho a data para o lançamento da versão final para os consumidores ainda não havia sido definida [MOTOGBRASIL, 2014]. Entretanto, o *kit* de desenvolvimento e a versão *preview* do sistema para desenvolvedores, já se encontra disponível, possibilitando que sejam testados os novos recursos e desenvolvidas ou adaptadas as aplicações existentes para o novo sistema. Na figura 2 é demonstrada uma comparação das telas de bloqueio da versão *KitKat* e L.

Figura 2 – Telas de bloqueio do *Android* 4.4 KitKat e *Android* L [GIZMODO, 2014].

De acordo com o site oficial do desenvolvedor *Android*, serão várias mudanças visuais e de desempenho nesta nova versão. Dentre as modificações são destacadas algumas como [ANDROID DEVELOPER, 2014]:

- **Um novo *design* da interface do usuário** - uma experiência consistente em móveis e na web com *Material Design*, o novo padrão visual;
- **Um novo *Runtime*** – o ART (*Android Runtime*) será o novo tempo de execução padrão na próxima versão;
- **Notificações melhoradas** - melhor controle sobre as notificações, simplificando o uso e a forma como são sincronizadas com dispositivos não portáteis;
- **Maior eficiência –** objetivo em tornar a energia da plataforma eficiente com o *Project* Volta, dando mais controle sobre o uso de recursos.

Além dessas, várias outras mudanças foram anunciadas e algumas até já se encontram documentadas no site do desenvolvedor *Android*, tornando possíveis os testes com o kit de desenvolvimento. E até que esteja disponível a versão final do sistema, muitas outras modificações poderão ocorrer.

3.2. Aplicações Java *web*

Uma aplicação Java *web* produz páginas interativas, contendo linguagem de marcação como HTML (*HyperText Markup Language* ou Linguagem de Marcação de Hipertexto), XML (*Extensible Markup Language* ou Linguagem de Marcação Extensível), dentre outras, assim como também conteúdo dinâmico composto por JSP (*JavaServer Pages*), *Servlets* e Java *Beans* [NETBEANS, 2014].

Para auxiliar nas tarefas de desenvolvimento destas aplicações, que podem muitas vezes ser repetidas, existem *frameworks* como o *JavaServer Faces* que oferecem bibliotecas com modelos de páginas, gerenciamento de sessão e visam a reutilização de código [CORDEIRO, 2014].

3.3. Java EE (Java *Enterprise Edition*)

As aplicações *web* na plataforma Java são baseadas na Java EE, que consiste em um grande conjunto de especificações. Existem diversas implementações, sendo que, os softwares que implementam apenas uma parte da

Java EE, atualmente são referenciados como "*application server web profile*" ou "*web server"* [CAELUM, 2014].

Os softwares que possuem o papel de oferecer às aplicações serviços de infraestrutura são chamados de servidores de aplicação. Alguns dos servidores mais conhecidos do mercado são [CAELUM, 2014]:

- GlassFish *Server Open Source Edition*;
- JBoss *Applicattion Server*.

3.3.1. GlassFish

O *GlassFish* é um servidor de aplicação, que suporta todas as especificações da API Java EE e linguagens dinâmicas, além de permitir aplicações corporativas portáveis, escaláveis e fáceis de integrar. Ele utiliza uma variante do Apache *Tomcat* (*Grizzly*) para manter essa eficiência [ORACLE, 2014].

3.3.2. JBoss

O servidor de aplicações JBoss é uma plataforma baseada em Java EE 5 utilizado para desenvolvimento e implantação de aplicativos empresariais. Este servidor suporta tanto APIs tradicionais quanto APIs do Java, podendo integrar o Apache *Tomcat* como seu contêiner *Web* e incluindo recursos para cache de dados, *clustering*, mensagens, transações e um serviço *Web* integrado [ORACLE, 2014].

3.3.3. Apache Tomcat

O Apache *Tomcat* é um Servlet *contêiner ou web server*, que consiste em um servidor que não dispõe de todos os recursos da Java EE, mas suporta a execução de tecnologias Java como Servlets, JSP e JSF para um ambiente *web*, sendo bastante indicado para aplicações de pequeno à médio porte [LUCKOW e MELO, 2010].

3.4. *JavaServer Faces*

Segundo LUCKOW e MELO [2010], *JavaServer Faces* (JSF) é a especificação para um *framework* de componentes para desenvolvimento *web* em Java. Este *framework* foi projetado com o objetivo de facilitar o desenvolvimento e a manutenção de aplicações executadas em um servidor de aplicações Java e processar as interfaces gráficas de volta a um cliente de destino [FRANCO, 2011].

A primeira versão do JSF enfrentou muita resistência por parte dos desenvolvedores, pelo fato de ser um *framework* baseado em componentes, e não em ações, como eram os principais *frameworks* da época. O lançamento da segunda versão teve como finalidade facilitar o seu uso com funções e desempenhos melhorados, oferecendo uma ferramenta produtiva de desenvolvimento de aplicações *web* [CORDEIRO, 2014].

O *JavaServer Faces* 2 tem como objetivo colaborar para que desenvolvedores possam construir aplicações *web* rapidamente, utilizando componentes reutilizáveis em uma página, conectando estes componentes a fontes de dados e gerenciando os eventos gerados pelo cliente no lado do servidor [JAVASERVER FACES, 2014].

O JSF possui recursos como HTML de formulário, tabelas, *layout*, conversão e validação de dados e eventos, interpretação de arquivos de configuração e interação com contêiner Java [GEARY e HORSTMANN, 2010].

Uma característica que se destaca é que, por fazer parte da especificação Java, ou seja, ser por padrão incluso na plataforma Java EE, não é necessário adicionar bibliotecas externas para utilizá-lo [ORACLE DOCS, 2014]. Outra vantagem relacionada a esta característica é a possibilidade de empresas investirem em desenvolvimento de componentes para o JSF. Podemos citar como exemplo de componentes: recurso de auto completar, calendários, *datagrids*, *drag & drop*, editor de textos, efeitos, dentre outros [LUCKOW e MELO, 2010].

Algumas das bibliotecas mais populares são:

- *RichFaces*, da JBoss (http://www.jboss.org/richfaces/);
- *ICEFaces*, da ICESoft (http://www.icefaces.org/);

- *PrimeFaces* (http://www.primefaces.org/).

Neste trabalho será abordada somente a *PrimeFaces*, devido a conhecimentos prévios sobre esta biblioteca.

3.4.1. *PrimeFaces*

O *PrimeFaces* é uma biblioteca de componentes desenvolvida para o *JavaServer Faces*. Possui código aberto e uma grande comunidade dedicada e ativa, que inclui diversos desenvolvedores. Seu principal objetivo é fazer com que o desenvolvedor crie uma camada de visualização, com uma aparência agradável para o usuário final, sem necessitar de dependências e configurações adicionais. Dentre alguns componentes disponíveis podem ser citados: gráficos, tabelas, caixas de mensagens, painéis, menus e *templates* [PRIMEFACES, 2014].

A figura 3 demonstra o componente *DataTable* da biblioteca *Primefaces*, sendo possível se ter uma noção do visual dos componentes da biblioteca.

Figura 3 – Componente *DataTable* do *PrimeFaces*. Adaptada de [RFAVERO, 2014].

Contando com mais de 90 componentes, o *PrimeFaces* é uma das mais completas bibliotecas, tendo sido uma das primeiras a estar totalmente convertida para o JSF 2.0 [LUCKOW e MELO, 2010]. No site oficial a documentação do *PrimeFaces* proporciona um material completo através de um guia de usuário que demonstra as definições de todos os componentes e exemplos de códigos utilizando-os.

3.5. *WebServices*

WebServices são softwares originados da Arquitetura Orientada a Serviços (SOA - *Service-Oriented Architecture*), os quais têm como objetivo padronizar o mecanismo de comunicação e permitir interoperabilidade de aplicações distintas, desenvolvidas em linguagens de programação diferentes e usando diferentes sistemas operacionais [RAMOS, 2012; NAGAPPAN et all, 2003].

3.5.1. *WebService* SOA

Um *WebService* em SOA é fundamentado em duas estruturas: o serviço e a descrição do serviço. O serviço é referente a um módulo de software, disponível para que seja utilizado por um cliente com acesso a rede, e a descrição do serviço consiste nos detalhes da implementação, os quais permitem o seu uso [RAMOS, 2012].

Os serviços são descritos em uma lista através de um contrato, podendo ser usados e reutilizados mantendo uma relação mínima de dependência (acoplamento fraco) e, além disso, podem ser compostos, ou seja, um serviço pode utilizar outros serviços [CERAMI, 2002].

A pilha de protocolos usados nos *WebServices* no padrão SOA pode ser dividida em quatro camadas [MORO et all, 2014]:

- **Busca e Descoberta –** usando protocolo UDDI (*Description, Discovery and Integration*), que consiste num protocolo que disponibiliza métodos padrões para a localização e publicação de informação do *WebService*;
- **WSDL** (*Web Services Description Language* ou Linguagem de Descrição de Serviços web) **–** Responsável pela descrição dos serviços;
- **Troca de mensagens –** usando protocolos XML-RPC (*Remote Procedure Call* ou Chamada de Procedimento Remoto), SOAP (*Simple Object Access Protocol* ou Protocolo Simples de Acesso a Objetos) e XML;
- **Transporte –** usando protocolos HTTP (*Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto), SMTP (*Simple Mail Transfer Protocol* ou Protocolo de Transferência de Correio Simples), FTP (*File*

Transfer Protocol ou Protocolo de Transferência de Arquivos), BEEP (*Blocks Extensible Exchange Protocol* ou Protocolo de Bloco Extensivo de Troca).

3.5.2. *WebService* REST

REST significa *Representational State Transfer* (Transferência de Estado Representativo), e é definido como um estilo arquitetural para a construção de sistemas distribuídos. Este termo teve origem na tese de doutorado de Roy Thomas Fielding, coautor de um dos protocolos mais utilizados do mundo, o HTTP [RAMOS, 2012].

Os *web services* baseados no conceito REST são denominados *RESTful* e surgiram como forma de simplificar o desenvolvimento de serviços *web*, evitando o excesso de padronização existente nos serviços SOAP, utilizando padrões já utilizados na web como HTTP, XML, JSON, URI, dentre outros, tornando assim, mais fácil a publicação e utilização dos serviços [SAUDATE, 2014].

O elemento base no REST é o recurso. Sendo assim, tudo é definido em termos de recursos, e estes são os dados a trafegarem pelo protocolo. Os recursos são representados por URI´s (*Universal Resource Identifier* ou Identificador Universal de Recurso), os quais servem para identificar qualquer artefato, como página web, figura, imagem ou vídeo [MORO et all, 2014].

Os dois principais fundamentos do REST são [RAMOS, 2012]:

- **Recurso de identificação por URI –** um serviço *web RESTful* expõe um conjunto de recursos identificados com endereçamento global que torne possível sua descoberta;
- **Interface Uniforme –** os recursos são manipulados por métodos do protocolo HTTP, de forma que os principais são: *GET*, *POST*, *PUT*, *DELETE*. Sendo *GET* utilizado para recuperar o estado de um recurso, *POST* para criar um novo estado de um recurso, *PUT* com a função de editar um recurso existente e *DELETE* remover.

É possível aprofundar nos fundamentos sobre o estilo REST, no entanto, para este trabalho, os descritos anteriormente são satisfatórios para a compreensão do cenário prático.

3.5.3. JSON e *Gson*

O JSON (*JavaScript Object Notation*) é um mecanismo ou formato de codificação/decodificação de valores para intercâmbio de dados. Possui uma sintaxe de alto nível, fácil de ser entendida. E por se significativamente mais simples em comparação com o XML, ela tem sido amplamente adotada [RAMOS, 2012].

Para se trabalhar com JSON pode se utilizar dentre outras a *Gson*, que é uma biblioteca da *Google* usada para converter objetos Java em suas representações em JSON [GSON, 2014].

3.5.4. *Framework* para *RESTful* em Java

Para realizar o desenvolvimento de *web services RESTful* na plataforma Java é possível utilizar *frameworks* que auxiliem nessa tarefa.

O *Jersey framework* é uma biblioteca que implementa e expande a especificação JAX-RS, que é o padrão em Java baseado em anotações, colaborando assim para a construção de *WebServices* REST. Sua versão mais atual é a 2.7 [JERSEY, 2014].

4. METODOLOGIA

Esta sessão tem como objetivo demonstrar como foram realizados os procedimentos técnicos do projeto. Para tanto, é necessário relatar os ambientes utilizados, tecnologias, ferramentas, assim como os resultados obtidos.

A seguir, na tabela 3, podem ser observadas as aplicações desenvolvidas com seus respectivos elementos, necessários para o funcionamento.

Tabela 3 – Ambientes utilizados e suas tecnologias.

Aplicação	Tecnologias	Servidor	Acesso a Dados	Frameworks / Bibliotecas
WebService RESTful	Java EE /JAX-RS	Apache TomCat 7	MySQL	Jersey, Gson
Aplicativo Android	Java Android	-	SQLite e Comunica-se com o WebService	Gson
Aplicação Web	Java EE	Apache TomCat 7	Comunica-se com o WebService	JSF 2.0, Gson e PrimeFaces 4

Com o intuito de manter o serviço com disponibilidade para testes e posteriormente para o uso em produção, a aplicação *web*, o *WeService RESTful* e o banco de dados foram publicados em um serviço de computação em nuvem, o AWS (*Amazon Web Services*).

Utilizando a AWS *Elastic Beanstalk*, uma ferramenta com interface *web* da *Amazon*, as tarefas de *upload, deploy* e gerenciamento das versões da aplicação *RESTful* foram bastante facilitadas.

Ao solicitar a criação de um novo ambiente nesta ferramenta, foi possível selecionar, dentre outras opções, uma predefinição com contêiner Apache *TomCat* versão 7.

Para o cenário pretendido, foi utilizada somente instâncias "t1.micro*"*, que equivale aproximadamente a um processador de 1.0 a 1.2 *Gigahertz* e 600 *Megabytes* de memória RAM (*Random Access Memory* ou Memória de Acesso Aleatório), a qual representa o servidor com *hardware* mais modesto disponível na AWS, normalmente usado em configurações iniciais.

No entanto, se necessário, é possível a qualquer momento adicionar mais instâncias com balanceamento de carga, alterá-las ou acrescentar várias outras ferramentas ou serviços, possibilitando assim aumentar o desempenho ou disponibilidade da aplicação. Na figura 4 é exibida a tela do AWS *Elastic Beanstalk* que demonstra que uma versão do *WebService RESTful* está implantada.

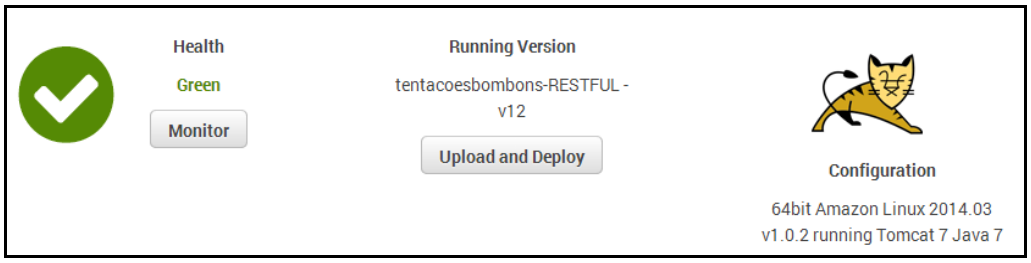


Figura 4 – Tela do Ambiente AWS *Elastik Beanstalk* [CONSOLE AMAZON, 2014].

No caso do banco de dados foi utilizada a ferramenta *Amazon* RDS (*Relational Database Service*), tendo sido criada uma instância do banco *MySQL* com armazenamento de 5 *Gigabytes*, que é o tamanho mínimo permitido para a criação de uma base de dados na AWS.

4.1. Requisitos

Para compreender a regra de negócio da empresa selecionada, na tabela 4 segue a listagem de requisitos funcionais e não funcionais do projeto:

Tabela 4 – Listagem dos requisitos funcionais e não funcionais.

Listagem de Requisitos		
Número	**Tipo**	**Descrição do Requisito**
001	Funcional	O sistema *mobile* deve permitir que usuários ativos realizem o *login* e deve impossibilitar que usuários inativos acessem o sistema.
002	Funcional	O sistema *mobile* deve permitir o cadastro, edição e desativação de uma Pessoa. Contendo: nome, sexo, data de nascimento, rg, cpf, endereço, telefone, *e-mail* e podendo ter status ativo ou inativo.
003	Funcional	O sistema *mobile* deve permitir o cadastro, edição e desativação de um Usuário. Contendo: pessoa, perfil, *login*, senha, confirmação de senha e podendo ter status ativo ou inativo.
004	Funcional	O sistema *mobile* deve permitir o cadastro, edição e desativação de um Cliente. Contendo: pessoa, empresa, setor e podendo ter status ativo ou inativo.
005	Funcional	O sistema *mobile* deve permitir o cadastro, edição e desativação de um Produto. Contendo: descrição, tipo do produto, valor e podendo ter status ativo ou inativo.
006	Funcional	O sistema *mobile* deve permitir realizar o lançamento de vendas. Contendo: cliente, itens da venda (produtos), tipo da venda, valor pago e valor de desconto.
007	Funcional	O sistema *web* deve permitir que usuários ativos realizem o *login* e deve impossibilitar que usuários inativos acessem o sistema.
008	Funcional	O sistema *web* deve permitir que usuários ativos com perfil "Cliente" realizem o *login* e visualizem o relatório de débitos, sendo necessário informar a data início e data final para que o relatório seja exibido.
009	Funcional	O sistema *web* deve permitir que usuários ativos com perfil "Vendedor" ou "Administrador" realizem o *login* e visualizem o relatório de devedores, sendo necessário informar a data início e data final para que o relatório seja exibido.
010	Funcional	O sistema *web* deve permitir que usuários ativos com perfil "Vendedor" ou "Administrador" realizem o *login* e visualizem o relatório de vendas, sendo necessário informar a data início e data final para que o relatório seja exibido.
011	Não-Funcional	O custo inicial do projeto não pode exceder R$500,00.
012	Não-Funcional	O custo mensal de serviços para manter o sistema funcional não pode exceder R$80,00.
013	Não-Funcional	A latência das requisições com o servidor de aplicação não pode ser superior a 300ms (milissegundos).
014	Não-Funcional	O sistema *mobile* deve ser desenvolvido para dispositivos com o sistema operacional *Android* da *Google*, tendo suporte para a versão da API 4.1 ou superior, pois a empresa já dispõe de aparelhos com estas características.

Tendo em vista os requisitos, devem ser observados os anexos 1, 2 e 3, que se tratam respectivamente do Diagrama de Casos de Uso, Diagrama de Classes e Diagrama de Entidade Relacional, para que seja possível uma visão ampla dos detalhes acerca das entidades da aplicação.

É necessário salientar que o Diagrama de Classes e o Diagrama de Casos de Uso foram criados se baseando no desenvolvimento do *WebService*, pois nas aplicações *web* e *mobile* nem todas as funcionalidades referentes aos recursos do *RESTful* estarão disponíveis, devido ao curto tempo disponível para o desenvolvimento do projeto.

4.2. Desenvolvimento das Aplicações

Para demonstrar como foi realizado o desenvolvimento, será exibida e descrita parte do código fonte, assim como algumas telas relativas aos principais cenários, tornando perceptível como foi obtida a comunicação entre as aplicações.

4.2.1. Aplicação *Android* Consumindo *WebService RESTful*

O aplicativo *Android* realiza a requisição HTTP ao *WebService RESTful*, o qual se comunica com o banco de dados, que por sua vez efetua a operação SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada). Após a execução da tarefa no banco, o *WebService* emite seu retorno, sendo este tratado na aplicação *Android* e utilizado para emitir o comportamento esperado, como por exemplo, exibir uma mensagem de sucesso ou falha ao usuário do sistema.

Ao realizar requisições HTTP em uma aplicação *Android*, recomenda-se que sejam utilizadas *AsyncTasks*. A *AsyncTask* é um mecanismo apropriado para dar *feedback* ao usuário durante a execução de alguma operação que demande tempo, colaborando assim, para que a interface não trave [PEREIRA e SILVA, 2009]. Na figura 5, é possível visualizar parte do código da classe "FinalizarVendaTask", a qual estende da classe pai *AsyncTask*, sendo o ciclo de vida deste mecanismo gerenciado dentro desta classe.

```
private class FinalizarVendaTask extends AsyncTask<Void, Void, ResponseWs> {
    private ResponseWs respostaWs = new ResponseWs();

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = ProgressDialog.show(LancamentoVendaActivity.this,
                getString(R.string.titulo_aguarde),
                getString(R.string.msg_finalizando_venda));
        progressDialog.show();
    }

    @Override
    protected ResponseWs doInBackground(Void... params) {
            return realizarVenda();
    }

    @Override
    protected void onPostExecute(ResponseWs result) {
        super.onPostExecute(result);
        progressDialog.dismiss();

        try {
            if (result != null) {
                if (respostaWs.isSucesso()) {
```

Figura 5 – *AsyncTask* responsável em realizar a requisição de lançamento de uma venda – Classe LancamentoVendaActivity.java.

No método "*doInBackground"* de uma classe *AsyncTask* é definida a tarefa a ser executada em segundo plano. Assim, neste caso, é chamado o método "realizarVenda" que com os dados informados pelo usuário, envia um objeto do tipo esperado para o método "lançarVenda" do seu *"controller"*, como é possível observar na figura 6.

```
try {
    respostaWs = vendaController.lancarVenda(venda);
} catch (Exception e) {
    Log.e(TAG, "Erro: " + e.getMessage());
    respostaWs.setMsg(getString(R.string.erro_servidor));
}
```

Figura 6 – Chamada do método "lancarVenda" da *controller* de Venda.

Visualizando o método "lancarVenda" na classe "VendaController", nota-se a chamada do "adicionarVenda", o qual retorna um código de resposta HTTP, sendo este tratado na *controller* para que seja enviada uma mensagem específica para a interface do usuário, como é possível acompanhar na figura 7.

```
public ResponseWs lancarVenda(Venda venda){
    ResponseWs responseWs = new ResponseWs();
    String codigoResponse = "";

    try {
        codigoResponse = vendaDao.adicionarVenda(venda);

        if(codigoResponse.equals("201") || codigoResponse.equals("200")){
            responseWs.setSucesso(true);
            responseWs.setMsg("Venda Finalizada com Sucesso!");
        }else{
            responseWs.setSucesso(false);
            responseWs.setMsg("Falha ao Finalizar a Venda!");
        }
    } catch (Exception e) {
        responseWs.setSucesso(false);
    }
    return responseWs;
}
```

Figura 7 – Método "lancarVenda" da classe VendaController.java.

No método "adicionarVenda", pertencente a classe de acesso a dados, a "VendaDAO", é composta a URL que fará a chamada ao *WebService* e também é criado o objeto da biblioteca *Gson*, o qual converte os dados para o formato JSON. Esses parâmetros adicionados ao tipo de método de requisição HTTP, neste caso o *POST*, são necessários para se realizar a chamada do método "httpPost". A figura 8 demonstra a codificação deste trecho.

```
public String adicionarVenda(Venda venda) throws Exception {
    String URL = ServiceUrl.getURL_SERVICE() + "vendas";
    Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd'T'HH:mm:ss").create();
    String vendaJSON = gson.toJson(venda);
    String codigoResponse = "";

    try {
        codigoResponse = conexao.httpPost(vendaJSON, URL, POST);
    } catch (Exception e) {
        codigoResponse = "500";
    }
    return codigoResponse;
}
```

Figura 8 – Método "adicionarVenda" da classe VendaDAO.java.

Para realizar a chamada das requisições do *WebService* foi criada uma classe útil nomeada de "FabricaConexao.java". Nesta classe o método "httpPost" tem o objetivo de atender tanto a requisições do tipo *POST*, quanto para o tipo *PUT*. São utilizados objetos nativos da API *Android* para realizar a conexões

HTTP e transmitir dados de entrada e saída. A seguir, na figura 9, observa-se o método “httpPost” completo.

```
public String httpPost(Object objeto, String stringURL, String metodo)
        throws IOException  {
    URL url = new URL(stringURL);
    String response = "";
    String output = "";

    HttpURLConnection conexao = (HttpURLConnection) url.openConnection();
    conexao.setRequestMethod(metodo);
    conexao.setDoOutput(true);
    conexao.setRequestProperty("Content-Type", "application/json");

    String input = objeto.toString();
    OutputStream os = conexao.getOutputStream();
    os.write(input.getBytes());
    os.flush();

    if (conexao.getResponseCode() != 200) {
        throw new RuntimeException("Erro HTTP: " + conexao.getResponseCode());
    }

    BufferedReader br = new BufferedReader(
            new InputStreamReader((conexao.getInputStream())));
    while ((output = br.readLine()) != null) {
        Log.i("TAG","Retorno response: " + output);
        response = output;
    }
    conexao.disconnect();
    return response;
}
```

Figura 9 – Método “httpPost” da classe FabricaConexao.java.

4.2.2. Analisando o *WebService RESTful*

Para analisar o código criado no *WebService* é interessante fazê-lo em ordem inversa ao que foi realizado no *Android*.

Primeiramente, é válido observar a classe “VendaDAO” que comunica-se com o banco de dados. Nesta classe existe o método “inserirVenda” que tem como função salvar os dados da venda no banco. Este método utiliza objetos da plataforma Java EE para realizar tarefas como: criar uma conexão com o banco de dados (*Connection*), execução de SQL (*PreparedStatement*) e retorno de transações (*ResultSet*). Na figura 10 é possível observar parte do método.

```
private int inserirVenda(Venda venda) {
    int result = 0;
    Connection conexao = null;
    PreparedStatement ps = null;
    ResultSet generatedKeys = null;

    try {
        conexao = FabricaConexao.getConexao();
        String sql = "INSERT INTO TB_VENDA "
                + "(id_usuario, id_cliente, id_tipo_venda, desconto, "
                + "dt_venda, dt_vencimento, valor_Total)"
                + " VALUES (?,?,?,?,?,?,?)";

        ps = conexao.prepareStatement(sql,PreparedStatement.RETURN_GENERATED_KEYS);
        ps.setInt(1, venda.getUsuario().getId());
        ps.setInt(2, venda.getCliente().getId());
        ps.setInt(3, venda.getTipoVenda().getId());
        ps.setDouble(4, venda.getDesconto());
        ps.setString(5, formatador.formataDataHora(venda.getDtVenda()));
        ps.setString(6, formatador.formataData(venda.getDtVencimento()));
        ps.setDouble(7, venda.getValorTotal());
        result = ps.executeUpdate();
```

Figura 10 – Método que lança uma nova venda no *WebService* – Classe DAO.

A próxima etapa é realizada na classe "VendaController", onde o resultado do método anterior é tratado. Tendo em vista que o método da classe "VendaDAO" deve retornar um valor inteiro, é verificado se este retorno é maior que zero para que seja constatada que a ação foi realizada com sucesso ou não. Na figura 11 observa-se este método.

```
  public boolean adicionaVenda(Venda venda){
      boolean sucesso = false;
      int resultado = vendaDao.adicionaVenda(venda);

      if(resultado > 0){
          sucesso = true;
      }
      return sucesso;
  }
```

Figura 11 – Método que adiciona uma nova venda no *WebService* – Classe VendaController.

A última classe a ser apresentada no processo de gravação da venda é a "VendaResource", a qual corresponde ao local onde são gerenciados os recursos REST do *WebService*. Nesta classe são visíveis os elementos da biblioteca *Jersey*, assim como o tratamento dos códigos de resposta HTTP.

A figura 12 exibe o código onde são percebidas as anotações utilizadas do *Jersey*, onde *@POST* representa o método HTTP, *@Consumes* o formato de dados que o método recebe e *@Produces* é referente ao formato que será retornado pelo método. Na variável "codigoHttp" é guardado o código que será retornado pelo método.

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
public String adicionaVenda(Venda venda) {
    boolean sucesso = vendaController.adicionaVenda(venda);
    String codigoHttp = "";

    if(sucesso){
        codigoHttp = "201"; //Criado
    }else{
        codigoHttp = "500"; //Erro de Servidor
    }
    return codigoHttp;
}
```

Figura 12 – Método que adiciona uma nova venda no *WebService* – Classe VendaResource.

Ao fim do procedimento, assim que a aplicação *Android* recebe o código de retorno do *WebService*, é exibida a mensagem de sucesso de gravação de uma venda. A seguir, na figura 13, é possível observar parte da tela de lançamento de vendas, sendo que esta tela completa e outras podem ser visualizadas no anexo 4.

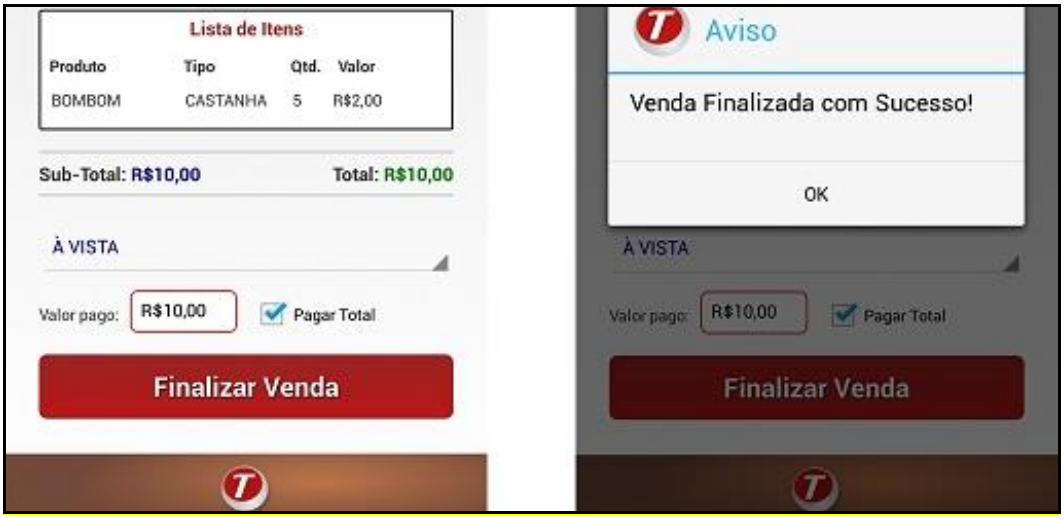


Figura 13 – Tela de Lançamento de Venda.

4.2.3. Autenticação na *web* utilizando *Phase Listener*

Para realizar a validação do *login* do usuário no sistema *web*, foi implementado um *PhaseListener*, que se trata de um objeto do JSF que tem a capacidade de interceptar uma requisição e realizar uma ação especifica.

O *PhaseListener* tem alguns métodos obrigatórios, no entanto, será demonstrado apenas o *afterPhase*, o qual tem a função de processar uma fase do ciclo de vida JSF quando já se completou. Na figura 14 é possível observar este método sendo usado com o objetivo de validar um usuário, de forma a recusar o acesso a uma página caso o usuário não seja autêntico.

```
@Override
public void afterPhase(PhaseEvent event) {
    FacesContext context = event.getFacesContext();
    ExternalContext external = context.getExternalContext();
    String path = context.getViewRoot().getViewId();

    if (!"/publico/usuario.xhtml".equals(path)) {
        if (external.getSessionMap().get(
                UsuarioBean.USUARIO_AUTENTICO) == null) {
            try {
                external.redirect(external.getRequestContextPath()
                        + USER_LOGIN_OUTCOME);
                context.renderResponse();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figura 14 – Classe AutenticationPhaseListener no sistema *web*.

A figura 15 demonstra a declaração da classe "AutenticationPhaseListener" na configuração do arquivo "faces-config.xml", com o intuito de marcar a classe responsável em escutar o ciclo de vida do JSF.

```
<lifecycle>
    <phase-listener>
        br.com.tentacoesweb.listener.AutenticationPhaseListener
    </phase-listener>
</lifecycle>
```

Figura 15 – Configuração no arquivo faces-config.xml no sistema *web*.

4.2.4. Consultando um relatório pelo ambiente *Web*

Uma venda realizada no sistema *Android* pode ser verificada somente pelo sistema *web*, pois pela restrição de tempo do projeto, foi estabelecido que nesta primeira versão não haveria o módulo de relatórios na aplicação móvel. A figura 16 mostra parte da tela do Relatório de Compras. A tela completa pode ser visualizada no Anexo 5.

Figura 16 – Tela de Relatório de Compras no sistema *web*.

Alguns componentes do JSF e *Primefaces* utilizados no projeto podem ser visualizados na figura 17, que apresenta parte do arquivo XHTML (*eXtensible Hypertext Markup Language*) relativo a página do Relatório de Compras.

```
<p:outputPanel>
    <h:panelGrid id="pnlComprasCliente" columns="6">
        <h:outputText value="Data início: " />
        <p:calendar id="dtInicio"
            value="#{relComprasClienteBean.rcc.dataInicio}" locale="de"
            navigator="true" pattern="dd/MM/yyyy" mask="true" />
        <h:outputText value="Data fim: " />
        <p:calendar id="dtFim"
            value="#{relComprasClienteBean.rcc.dataFim}" locale="de"
            navigator="true" pattern="dd/MM/yyyy" mask="true" />
        <p:commandButton value="Consultar"
            action="#{relComprasClienteBean.relComprasCliente}"
            update="tabelaComprasCliente" ajax="false" />
        <h:outputText
            value="Saldo Atual: R$#{relComprasClienteBean.saldoCliente}"
            id="saldoAtual" styleClass="saldoAtual" />
    </h:panelGrid>
</p:outputPanel>
```

Figura 17 – Tela de Relatório de Compras no sistema *web*.

Um dos diferenciais do JSF que percebesse no código apresentado, é que o desenvolvedor pode trabalhar com o código referente à camada de visualização sem que necessite misturar lógica de programação com a linguagem de marcação, fato este que contribui positivamente na agilidade em projetos onde há desenvolvedores de *front-end* e *back-end* envolvidos.

No botão "Consultar" da tela de Relatório de Compras é chamado o método "relComprasCliente" da classe "RelComprasClienteBean". A figura 18 exibe este método.

```
public String relComprasCliente() {
    try {
        dataInicio = formatador.formataData(rcc.getDataInicio());
        dataFim = formatador.formataData(rcc.getDataFim());
    } catch (Exception e) {
        System.out.println("Erro: " + e);
    }
    id = usuarioBean.getUsuario().getPessoa().getId().toString();
    relCompras = rcn.gerarRelatorio(id, dataInicio, dataFim);

    if (relCompras.isEmpty()) {
        FacesUtil.addMessage(FacesMessage.SEVERITY_WARN, "Aviso",
                "Dados não encontrados");
        return null;
    } else {
        valorPagoPeriodo();
        saldoCliente();
        somaVenda();
        return "list_rel_compras_cliente";
    }
}
```

Figura 18 – Classe RelComprasClienteBean no sistema *web*.

Uma classe *Bean* num projeto Java *web* tem a função de prover acesso a objetos, auxiliar na comunicação entre as camadas de visualização e regra de negócios, e ainda gerenciar o ciclo de vida dos objetos. Portanto, para possibilitar que o componente *DataTable* utilizado na tela do relatório mantivesse os objetos, mesmo ao utilizar a paginação, foi realizada a anotação *@SessionScope* na classe "RelComprasClienteBean" [FRANCO, 2011]. Ainda por meio desta classe é chamado o método "gerarRelatorio" da classe de regra de negócios, a "RelComprasClienteRN".

A figura 19 exibe a implementação deste método, o qual se comunica com a classe DAO pelo método “relCompras”.

```
public List<RelatorioComprasCliente> gerarRelatorio(
        String id, String dataInicio, String dataFim) {
    List<RelatorioComprasCliente> relatorio =
            new ArrayList<RelatorioComprasCliente>();

    relatorio = rccDAO.relCompras(id,dataInicio,dataFim);
    return relatorio;
}
```

Figura 19 – Classe RelComprasClienteRN no sistema *web*.

Por fim, analisando o método “relCompras” da classe “RelComprasClienteDAO”, o qual é exibido na figura 20, nota-se a montagem de parte da URL que realiza a requisição com o *WebService* e também é possível observar a utilização da biblioteca *Gson*.

```
public List<RelatorioComprasCliente> relCompras(String id,
        String dataInicio, String dataFim) {

    List<RelatorioComprasCliente> relatorio =
            new ArrayList<RelatorioComprasCliente>();
    try {
        url = "relatorioComprasCliente" + "/"
                + id + "/" + dataInicio + "/" + dataFim;
        wsRelComprasCliente = fc.conexaoWs(url);

        Gson gson = new Gson();
        JsonParser jsonParser = new JsonParser();
        JsonArray relComprasArray =
                jsonParser.parse(wsRelComprasCliente).getAsJsonArray();

        for (JsonElement jsonRelCompras : relComprasArray) {
            RelatorioComprasCliente rcc = gson.fromJson(jsonRelCompras,
                    RelatorioComprasCliente.class);
            relatorio.add(rcc);
        }
    } catch (Exception e) {
        System.out.println("Erro: " + e);
        return null;
    }
    return relatorio;
}
```

Figura 20 – Classe RelComprasClienteDAO no sistema *web*.

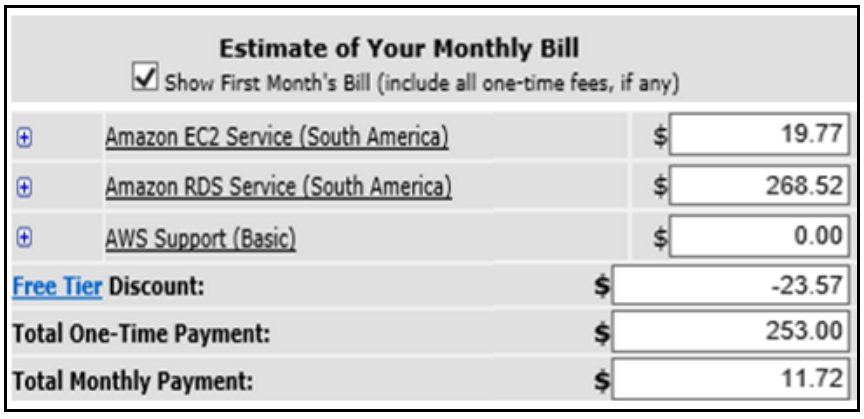
Na aplicação *web* também foi criada uma classe "FabricaConexao.java", semelhante a implementação realizada no *Android*, a qual tem como objetivo realizar a requisição com o *WebService*. Após a aplicação receber o retorno desta requisição, os dados são tratados e ao término entregues a interface do usuário para que sejam exibidos.

4.3. Estimativa de Custos

As aplicações foram desenvolvidas pela equipe autora deste documento, sendo assim, o custo inicial de desenvolvimento é, portanto equivalente a zero, pois o tempo envolvido é relativo ao trabalho. Sabendo que um dos integrantes da equipe, é também sócio proprietário da empresa, o custo de manutenção e melhorias será absorvido.

No entanto, como foi definido utilizar um servidor locado em *datacenter*, haverá um custo mensal para este item.

Se locado na AWS (*Amazon Web Services*), com instâncias em São Paulo reservadas por 3 anos, o custo mensal será de 11,72 dólares, equivalente a R$26,06 (cotado em 02/07/2014). Na figura 21 temos a simulação de valores.



Estimate of Your Monthly Bill

Show First Month's Bill (include all one-time fees, if any)

⊞	Amazon EC2 Service (South America)	$ 19.77
⊞	Amazon RDS Service (South America)	$ 268.52
⊞	AWS Support (Basic)	$ 0.00
Free Tier Discount:		$ -23.57
Total One-Time Payment:		$ 253.00
Total Monthly Payment:		$ 11.72

Figura 21 – Cálculo de valor mensal do serviço na AWS [AMAZONWS, 2014].

No entanto, este serviço é gratuito por um ano, ocasionando em economia e gerando tempo para que sejam pesquisados mais serviços equivalentes e analisadas as características e viabilidade de cada *datacenter*.

4.4. Conclusão

A integração entre sistemas tornou-se ao decorrer dos anos mais que um diferencial, um verdadeiro requisito essencial. Da mesma forma, a mobilidade vem adquirindo espaço e se consolidando como parte do cotidiano das pessoas, fazendo com que fiquem totalmente dependentes de dispositivos e seus aplicativos.

No meio corporativo, deve-se sempre analisar o modelo da empresa para que seja decidido se é prudente investir em uma tecnologia ou não.

O projeto proposto para a empresa foco deste trabalho demonstrou-se viável e oportuno, pois com um sistema estruturado que ofereça integração e mobilidade, os processos que se referem aos cadastros e as vendas tendem a se tornar mais eficientes.

É evidente que os maiores resultados serão obtidos em longo prazo, pois o sistema ainda não está pronto, no entanto, com o potencial que demonstra, quando estiver em produção será uma ferramenta de grande utilidade para a empresa.

A aplicação móvel e *web* tendem a evoluir, sendo que, a intenção é que posteriormente ocorram melhorias e seja desenvolvido um módulo para o cliente no *Android*, o qual permitirá que este visualize os relatórios de forma nativa em seu dispositivo, realize pedidos e solicite orçamentos.

Com o modelo de *WebService RESTful* construído é possível que se trabalhe de forma incremental, sempre adicionando novos recursos e ainda com o benefício de atender a qualquer outro tipo de aplicação futura.

No processo de desenvolvimento algumas dificuldades foram encontradas, relativas ao conhecimento das tecnologias ou pela restrição de tempo do projeto. Contudo, estes imprevistos foram superados buscando ajuda de profissionais da área, pesquisando soluções, trabalhando em grupo e realizando decisões de projeto, como a remoção de algumas funções extras almejadas na concepção do projeto, colaborando dessa forma para que os requisitos definidos fossem cumpridos.

Por fim, as tecnologias escolhidas para o projeto contribuíram para que o trabalho resultasse em vasto aprendizado, experiências foram adquiridas por todos

os integrantes do grupo, e ainda, os requisitos do projeto puderam ser alcançados de forma satisfatória.

REFERÊNCIA BIBLIOGRÁFICA

AMAZONWS. **Amazon WebService – Simple Monthly Calculator**. Disponível na internet: http://calculator.s3.amazonaws.com/index.html. 02 de julho 2014.

ANDROID DEVELOPER. **Android L Developer Preview**. Disponível na http://developer.android.com/preview/index.html. 01 de junho 2014.

CAELUM. **Apostila Java para Desenvolvimento Web – Servlet Container**. Disponível na internet: http://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/#3-4-servlet-container. 30 de junho 2014.

CERAMI, Ethan. **Web Services Essentials**. 1.ed. O'Reilly Media, 2002.

CONSOLE AMAZON. **Amazon WebService – Console Elastic Beanstalk**. Disponível na internet: https://console.aws.amazon.com/elasticbeanstalk. 02 de julho 2014.

CORDEIRO, Gilliard. **Aplicações Java para web com JSF e JPA.** 1. ed. São Paulo: Casa do Código, 2014.

FRANCO, Rebeca Such Tobias. **Estudo comparativo entre frameworks Java para desenvolvimento de aplicações web: JSF 2.0, Grails e Spring Web MVC**. Disponível na internet. http://repositorio.roca.utfpr.edu.br/jspui/handle/1/492. 23 abril 2014.

GEARY, David; HORSTMANN, Cay. **Core JavaServer Faces.** 3. ed. Michigan: Prentice Hall, 2010.

GIZMODO. **Android L: tudo o que você precisa saber**. Disponível na internet: http://gizmodo.uol.com.br/android-l-oficial/. 01 de junho 2014.

GSMARENA. **LG NEXUS 5 pictures**. Disponível na internet: http://www.gsmarena.com/lg_nexus_5-pictures-5705.php. 01 de julho 2014.

GSMARENA. **T-Mobile G1 pictures**. Disponível na internet: http://www.gsmarena.com/t_mobile_g1-pictures-2533.php. 01 de julho 2014.

GSON. **Gson User Guide**. Disponível na internet. https://sites.google.com/site/gson/gson-user-guide. 23 abril 2014.

JAVASERVER FACES. **Mojarra JvaServer Faces**. Disponível na internet. https://javaserverfaces.java.net/. 30 junho 2014.

JERSEY. **RESTful Web Services in Java**. Disponível na internet. https://jersey.java.net/. 19 abril 2014.

LECHETA, Ricardo. **Google Android – Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 2. ed. rev. e ampl. São Paulo: Novatec Editora, 2010.

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para a Web.** 1. ed. São Paulo: Novatec Editora, 2010.

MEDNIEKS, Zigurd; DORNIN, Laird; MEIKE, G. Blake; NAKAMURA, Masumi. **Programando o Android.** 2. ed. São Paulo: Novatec Editora, 2012.

MORO, Tharcis Dal; DORNELES, Carina; REBONATTO, Marcelo Trindade. **Web services WS-* versus Web Services REST.** Disponível na internet: www.seer.ufrgs.br/reic/article/download/22140/12928. 29 de junho 2014.

MOTOGBRASIL. **Android KitKat x Android L: comparação visual**. Disponível na internet: http://motogbrasil.com.br/android-kitkat-x-android-l-comparacao-visual/. 30 de junho 2014.

NAGAPPAN, Ramesh; SKOCZYLA, Robert; SRIGANESH, Patel. **Developing Java Web Services: Architecting and Developing Secure Web Services Using Java.** 1. ed. Indianopolis: Wiley Publishing Inc., 2003.

NETBEANS. **Introdução ao JavaServer Faces 2.x**. Disponível na internet. https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html. 17 abril 2014.

NETBEANS. **Trilha do Aprendizado do Java EE e Java Web**. Disponível na internet. https://netbeans.org/kb/trails/java-ee_pt_BR.html. 30 abril 2014.

ORACLE DOCS. **The Java EE 5 Tutorial** Disponível na internet. http://docs.oracle.com/javaee/5/tutorial/doc/bnaph.html. 30 junho 2014.

ORACLE. **Comparing Oracle GlassFish Server and JBoss: Which Application Server Is Right for You?** Disponível na internet. http://www.oracle.com/us/products/middleware/application-server/oracle-glassfish-server/comparing-glassfish-jboss-wp-117118.pdf. 17 abril 2014.

PEREIRA, Lúcio Camilo Oliva; SILVA, Michel Lourenço. **Android para desenvolvedores.** 1. ed. Rio de Janeiro: Brasport, 2009.

PRIMEFACES. **PrimeFaces User´s Guide 4.0**. Disponível na internet. http://primefaces.googlecode.com/files/primefaces_users_guide_4_0_edtn2.pdf. 22 de abril de 2014.

RABELLO, Ramon Ribeiro. **Android: um novo paradigma de desenvolvimento móvel**. Disponível na internet: http://www.cesar.org.br/site/files/file/WM18_Android.pdf. 29 de junho 2014.

RAMOS, Ricardo Ramos de. **Avaliação de manutenibilidade entre as abordagens de web services RESTful e SOAP-WSDL**, 2012. Disponível na internet. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-24072012-164751/pt-br.php. 19 abril 2014.

RFAVERO, Blogspot. **Utilização de Java + NetBeans e Frameworks diversos para desenvolvimento de portais**. Disponível na internet: http://www.rfavero.blogspot.com.br/2013/12/destaque-highlight-do-texto-filtrado-na.html. 30 de junho 2014.

SAUDATE, Alexandre. **REST – Construa API´s inteligentes de maneira simples.** 1. ed. São Paulo: Casa do Código, 2014.