Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

PAULO VITOR PAES ANDRADE

IMPLEMENTAÇÃO DE UM LANÇADOR UTILIZANDO WEB SERVICE REST

Assis/SP
2016

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

PAULO VITOR PAES ANDRADE

IMPLEMENTAÇÃO DE UM LANÇADOR UTILIZANDO WEB SERVICE REST

Projeto de pesquisa apresentado ao curso de Bacharelado em Ciências da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando : Paulo Vitor Paes Andrade
Orientador : Prof. Douglas Sanches da Cunha

Assis/SP
2016

FICHA CATALOGRÁFICA

ANDRADE, Paulo Vitor Paes.
Implementação de um lançador utilizando Web Service REST / Paulo Vitor Paes Andrade. Fundação Educacional do Município de Assis – FEMA – Assis, 2016.
42 páginas.

1. Web Service. 2. Interoperabilidade. 3.Java Web. 4.REST

CDD: 001.6
Biblioteca da FEMA

IMPLEMENTAÇÃO DE UM LANÇADOR UTILIZANDO WEB SERVICE REST

PAULO VITOR PAES ANDRADE

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Bacharelado em Ciências da Computação, avaliado pela seguinte comissão examinadora:

Orientador: _____
Prof. Douglas Sanches da Cunha

Examinador: _____
Prof. Diomara Martins Reigoto Barros

Assis/SP
2016

DEDICATÓRIA

Dedico este trabalho à minha família que sempre me apoiou em minhas decisões, e sempre esteve ao meu lado nas horas mais difíceis. Também dedico à Marcela de Souza Paião minha namorada, que também sempre está ao meu lado, caminhando e evoluindo juntos, com quem quero construir uma família.

AGRADECIMENTOS

À minha mãe Roberta de Fátima Paes que sempre me apoiou e a cada dificuldade me incentivava cada vez mais, à minha avó Ruth de Souza Andrade que nunca me desamparou e sempre que pode supri minhas necessidades. Às minhas tias Sandra Cristina Andrade Ferezin e Edna Maria de Andrade que sempre acreditaram no meu potencial e sempre me incentivaram em todos os momentos de minha vida. À Instituição da FEMA que procura sempre trazer palestras, e inovações para os alunos, e a semana dedicada ao curso de Ciência da Computação que sempre traz novidades e conhecimento para os alunos. Ao Mestre Douglas Sanches da Cunha pela oportunidade de ser seu orientando e a ajuda na elaboração do presente trabalho.

À uma mulher muito especial que sempre me incentivou, Marcela de Souza Paião, que sempre teve paciência quando por algum trabalho ou prova, não podíamos ficar juntos. E a todas as pessoas que contribuíram para minha formação direta ou indiretamente, o meu muito obrigado.

“A menos que modifiquemos à nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”.

(Albert Einstein)

RESUMO

Há um tempo atrás, os sistemas eram construídos para serem executados em máquinas especificas e eram totalmente acoplados, isso com o tempo e a evolução dos computadores, dos celulares e da internet, teve que ser revisto pois os usuários queriam mais mobilidade e mais integração entre sistemas. Com isso teve o surgimento dos sistemas distribuídos e desacoplados, por exemplo os WEB SERVICES que são sistemas capazes de distribuir informações independente da linguagem utilizada pelo sistema que consumirá essas informações, proporcionando a interoperabilidade entre sistemas.

O uso dos WEB SERVICES veio se popularizando ao decorrer do tempo, e assim também ocorreu a evolução dos mesmos. O tema tratado no presente trabalho é sobre uma destas evoluções, o REST. O WEB SERVICE REST é uma metodologia arquitetural para o desenvolvimento de um WEB SERVICE, suas características prezam pela simplicidade na troca de informações e pela comunicação rápida, fácil e segura entre sistemas.

Palavras-chave: *WEB SERVICE*, REST, Interoperabilidade, Sistemas desacoplados.

ABSTRACT

Not much time ago, systems were designed to run in specific machines and were also totally coupled. But then, with the evolution of computers, cellphones and internet, this had to be reviewed as the users wanted more mobility and integration of the systems. Due to this, the distributed and uncoupled systems came up, the WEB SERVICES for instance. These ones are systems capable of distributing information regardless of the language operated by the system that will consume it, what provides the interoperability between systems.

The use of WEB SERVICES started to become popular with time, and their evolution also occurred. The subject addressed in this paper is about one of the tools that make part of what is here called evolution, the REST. The WEB SERVICE REST is a architectural methodology for the development of a WEB SERVICE, its characteristics value the simplicity on the exchange of information and a fast, easy and safe communication between systems.

Key-words: WEB SERVICE, REST, Interoperability, Uncoupled systems.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Os *WEB SERVICES* surgiram devido aos grandes avanços tecnológicos, a popularização da internet, o surgimento de aplicações distribuídas e a necessidade do compartilhamento de informações.

No Brasil os *WEB SERVICES* são utilizados para a emissão de NF-e (Nota Fiscal Eletrônica), NFC-e (Nota Fiscal de Consumidor Eletrônica), NFS-e (Nota Fiscal de Serviço Eletrônica). No caso da NF-e e da NFC-e o *WEB SERVICE* é disponibilizado por estado, já na NFS-e cada município possui seu próprio WEB SERVICE.

O problema de interoperabilidade entre sistemas é sanado pelos *WEB SERVICES*, pois ele é capaz de ser consumido pelos mais diversos tipos de sistema, sem se preocupar com linguagem e plataforma aplicada, basta as informações serem enviadas no formato padrão reconhecido pelo *WEB SERVICE*.

As principais estruturas de dados reconhecidas pelos *WEB SERVICES* são: o XML, o SOAP, o WSDL e o UDDI. Estes padrões de dados gera uma melhor troca de informações entre o sistema e o *WEB SERVICE*, acabando com os problemas de comunicação entre aplicações distribuídas.

Segundo Menéndez (2002), há uma definição bastante simples para um *WEB SERVICE*: “É uma aplicação que aceita solicitações de outros sistemas através da Internet”.

O que há de mais novo no mundo dos *WEB SERVICES* é o Amazon Cognito, ele fornece identidade do usuário e sincronização de dados simples como, preferências do usuário e status dos jogos ambos sincronizados nos dispositivos.

Uma falha dos *WEB SERVICES* é na segurança das informações na troca de informações entre o sistema e o *WEB SERVICE*. O *man-in-the-middle* é uma falha onde um atacante consegue interceptar as informações entre os sistemas e modifica-las e também bloquear parte das informações.

1.1. OBJETIVO

O objetivo geral deste projeto é desenvolver um sistema que suas informações serão providas exclusivamente de um *WEB SERVICE*, onde ele fará a manutenção das informações na base de dados.

Tornando-se uma aplicação segura e a *WEB SERVICE* pode ser utilizado por outras aplicações proporcionando a interoperabilidade entre sistemas.

O objetivo específico deste projeto é desenvolver um lançador que terá como única estrutura o *front-end*, onde será implementada somente a parte visual do projeto. O *WEB SERVICE* tomará a arquitetura do *back-end*, onde será implementado todo o controle das informações contidas no banco de dados.

1.2. JUSTIFICATIVA

Com o crescimento do uso de *WEB SERVICES* em diversas áreas, a segurança proporcionada e a interoperabilidade entre qualquer tipo de aplicação. Muitas empresas utilizam de *WEB SERVICE* para a comunicação e o compartilhamento de várias informações entre diversos sistemas dentro da empresa.

Define-se uma importante área para ser pesquisada e trabalhada, com testes e falhas de segurança para serem resolvidas, uma área de pesquisa e futuro promissor.

1.3. MOTIVAÇÃO

A motivação para o desenvolvimento deste trabalho consiste na possibilidade de contribuir com a melhoria da segurança dos dados de uma aplicação e no conceito de que várias outras aplicações possam consumir os dados cedidos pela *WEB SERVICE*.

A motivação é, também poder estudar e aprender mais sobre *WEB SERVICE* que é algo ultimamente muito utilizado, principalmente em empresas que possuem mais de um sistema e a necessidade de integrá-los. Uma área onde o mercado de trabalho é muito promissor para a carreira de um profissional de TI.

Finalmente, uma outra motivação é poder aprender mais sobre Java que é uma das linguagens de programação mais utilizadas no mundo, onde o mercado também é muito promissor, pois grandes empresas desenvolvem em Java, aplicações web e desktop.

1.4. PERSPECTIVAS DE CONTRIBUIÇÃO

Na conclusão do projeto, proporcionará um sistema em formato de lançador, com funcionalidades de *login*, gerenciamento de usuários, gerenciamento de permissões de usuários. Também disponibilizando uma *WEB SERVICE* parcialmente genérica para consumo de informações, onde qualquer outro sistema que necessite das mesmas informações possa consumir esta *WEB SERVICE*.

1.5. METODOLOGIA DE PESQUISA

A proposta e objetivos deste trabalho acadêmico serão alcançados por meio de pesquisas teóricas, de forma a adquirir os conhecimentos necessários por meio da leitura de artigos científicos, monografias, teses, guias práticos e técnicos, livros e fontes digitais confiáveis, tornando possível a elaboração e o desenvolvimento de um sistema de tipo lançador com integração de *WEB SERVICE* REST.

Serão realizados estudos de tecnologias Java para o desenvolvimento do lançador, como linguagem Java, Spring, JPA, Bootstrap e Hibernate. De forma a conhecer e aprender cada uma das ferramentas e utiliza-las da melhor maneira.

Posteriormente será realizado um estudo aprofundado sobre *WEB SERVICE*, de forma conhecer e aprender suas formas de comunicação, o consumo e disponibilidade das informações, arquiteturas e padrões de linguagem.

Com o conhecimento pleno das tecnologias, será realizado o desenvolvimento do lançador e do *WEB SERVICE* para a comunicação de ambos os lados. Onde o lançador fará requisições ao *WEB SERVICE* que terá o papel de fazer a persistência e a busca das informações no banco de dados.

1.6. ESTRUTURA DO TRABALHO

Este trabalho será estruturado nas seguintes partes:

O capítulo 1 tem como título Introdução, nele é apresentado uma introdução sobre o assunto, objetivos, justificativa, motivação para a realização desse projeto.

O capítulo 2 tem como título *WEB SERVICE*, nele é apresentado uma introdução sobre *WEB SERVICE*, seus padrões de linguagem, comunicação, interoperabilidade, benefícios e desafios.

O capítulo 3 tem como título Java, nele é apresentado uma introdução sobre Java, as tecnologias utilizadas no projeto e o desenvolvimento do lançador.

O capítulo 4 tem como título REST, uma introdução sobre REST, suas características, os tipos de linguagem utilizados, recursos, representações, métodos e um comparativo com SOAP.

O capítulo 5 tem como título Proposta do Trabalho, nele é apresentada a proposta do trabalho, informando o que as finalidades do projeto.

O capítulo 6 tem como título Diagramas e Casos de Uso, nele é apresentado diagramas técnicos e estudos de caso de uso.

O capítulo 7 tem como título Considerações Finais e Trabalhos Futuros, nele é apresentado as considerações do trabalho e propostas de trabalhos futuros.

2. WEB SERVICE

Neste capítulo será apresentado uma introdução sobre *WEB SERVICE*, descrevendo o seu surgimento, funcionamento da plataforma, onde se aplica em uma sistema e sua arquitetura.

Um *WEB SERVICE* é basicamente um serviço disponibilizado na internet, que pode suprir necessidades de vários sistemas diferentes utilizando uma linguagem padrão, a figura 1 ilustra uma comunicação básica entre sistemas utilizando *WEB SERVICE*.

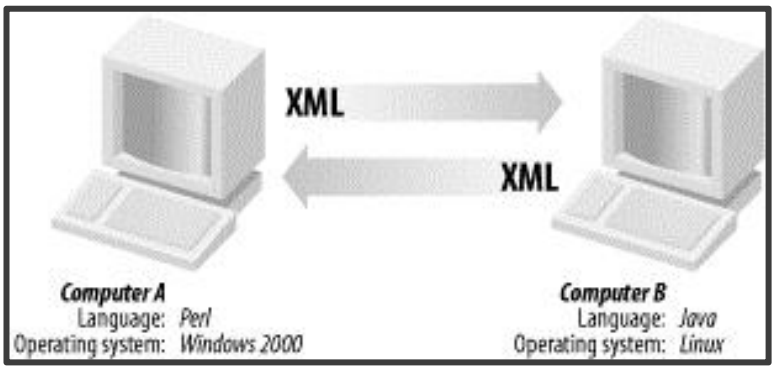

Figura 1 – Comunicação básica de um *WEB SERVICE.*

No ano de 2002, a W3C (*World Wide Web Consortium*) aceitou a implantação de um padrão de mensagem o SOAP (*Simple Object Access Protocol*), cujo o formato baseado em XML, que em seu conteúdo possui um conjunto de *<tags>* para a estruturação das informações contidas no arquivo, sendo mais consistente e imune de falhas, permitindo a troca de dados mais robusta como "*Arrays*".

Criou-se uma estrutura de comunicação e transmissão de informações entre serviços e aplicações por meio de HTTP. Fazendo com que as tecnologias não ficassem presas ao fabricante podendo se comunicar em um formato padrão, o SOAP proporcionou uma alternativa sobre os protocolos já existes como CORBA e DCOM.

No ano seguinte foram criadas tecnologias para suplementar o SOAP, tais como o WSDL que serve como uma nova implementação de XML, para descrever a interface dos *WEB SERVICES*, funcionando como uma "*TypeLibrary*" (Biblioteca de tipos) usada para a validação das chamadas dos métodos. E a especificação UDDI (*Universal Description Descovery and Integration*), que forneceu um mecanismo padrão de descoberta dinâmica de descrições de serviços. E assim a primeira plataforma de *WEB SERVICE* estava criada, a figura 2 ilustra as relações da primeira plataforma de *WEB SERVICE*.

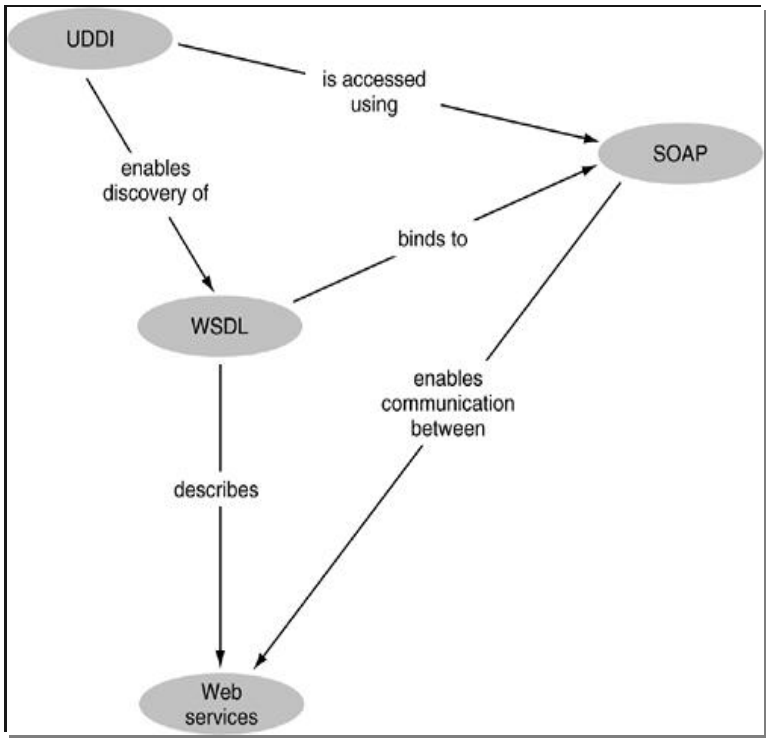

Figura 2 – Relações da Primeira plataforma de *WEB SERVICE* estabelecida.

Indicando as Relações da figura anterior:

- *Is Acessed Used*: é acessado utilizando;
- *Enables Discovery of*: permite a descoberta de;
- *Describes*: descreve;
- *Enables Communication Between*: permite comunicação entre;
- *Binds to*: ligação para.

Assim a popularização dos *WEB SERVICES* teve um ritmo relevante, adotado por vendedores e fornecedores. E teve sua importância na ampla indústria de

desenvolvimento de projetos principalmente nos orientados a serviços. Isto levou a criação da segunda geração de especificação de *WEB SERVICE*.

WEB SERVICE é uma plataforma orientada a serviços onde uma aplicação disponibiliza métodos que reproduzem ações de diversos tipos como, persistência em banco de dados, cálculos, entre outros. Onde possibilita que outras aplicações das mais variadas formas, como aplicações web, desktop, mobile, independente da linguagem, a única coisa que elas precisam ter em comum é uma linguagem padrão de comunicação entre o *WEB SERVICE* e a aplicação que utilizará o método disponibilizado, a figura 3 ilustra uma representação lógica de uma arquitetura orientada a serviços que os *WEB SERVICES* fazem parte.

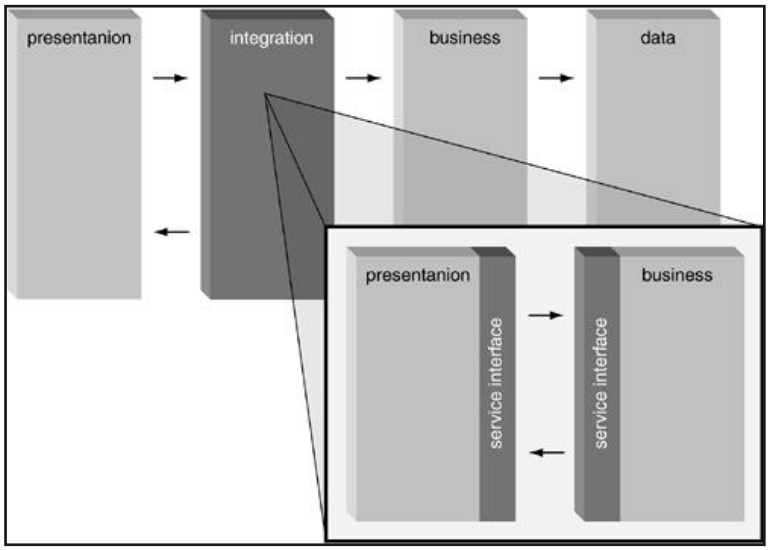

Figura 3 – Representação lógica de uma Arquitetura Orientada a Serviços.

Indicando as Fases da figura anterior:

- *Presentation*: apresentação;
- *Integration*: integração;
- *Business*: negócio;
- *Data*: dados;
- *Service interface*: interface de serviço.

Nesta representação lógica a apresentação seria a aplicação que se comunicaria com o *WEB SERVICE* fazendo uma solicitação, a integração seria o

próprio *WEB SERVICE* que estaria recebendo esta solicitação da aplicação, o negócio seria as regras de negócio que aplicadas no sistema, e os dados seria as informações contidas na base de dados.

A implementação de *WEB SERVICE* em uma empresa prove vários benefícios como, agilidade, reuso da infraestrutura, redução de custos, redução da dependência tecnológica, processo de desenvolvimento mais eficiente, mitigação de riscos e sobrevida para sistemas legados. Também extinguira a duplicidade de códigos dentro do setor de desenvolvimento, onde aplicações que precisem de uma mesma funcionalidade bastam consumirem do mesmo *WEB SERVICE* que disponibiliza a funcionalidade.

Neste capítulo é possível concluir que o surgimento dos *WEB SERVICES* foi e é muito importante para a área de tecnologia, proporcionando uma maior interoperabilidade entre sistemas, um desacoplamento total de um sistema, possibilitando que os processos mais complexos de uma aplicação sejam realizados no servidor, poupando hardware do dispositivo que pode ter o poder de processamento reduzido.

3. JAVA, E SUAS TECNOLOGIAS

Neste capítulo será apresentado uma introdução a tecnologia Java, apontando o surgimento, a plataforma e as tecnologias utilizadas no trabalho como Spring, JPA e Bootstrap.

O Java é umas das principais linguagens de programação utilizadas hoje em dia, e possui uma grande quantidade de tecnologias que auxiliam no desenvolvimento de aplicações simples e complexas.

3.1. O QUE É JAVA?

Criada pela Sun Microsystems a partir de um time para desenvolver inovações tecnológicas liderado por James Gosling (conhecido como pai do Java). A ideia principal era criar um interpretador que facilitaria a reescrita de software para aparelhos eletrônicos e pequenos dispositivos, como vídeo cassete, televisão e aparelhos de TV a cabo. Mas esta ideia não evoluiu muito por falta de investimento de outras empresas.

Com a evolução da web, a Sun percebeu que poderia usar a ideia do Java para rodar em pequenas aplicações no *browser*, foi aí que o Java 1.0 foi lançado para transformar o *browser* em apenas um cliente magro (*thin client* ou terminal burro) que possa realizar operações avançadas e não somente renderizar HTML.

Em 2009 a Oracle comprou a Sun, fortalecendo a marca. E junto com a IBM, fizeram grandes negócios e investimentos através do uso da plataforma Java. Hoje a linguagem Java domina os sistemas operacionais de pequenos dispositivos e de telefones móveis somando em mais de 2.5 bilhões de dispositivos compatíveis.

A linguagem Java é uma das mais utilizadas na área de desenvolvimento de softwares tanto para web quanto para aplicações desktop, ela é uma linguagem orientada a objetos, com foco em aplicações de médio e grande porte, no Java pode ser mais complicado e trabalhoso montar a primeira versão

da aplicação, mas após esta fase, alterações e crescimento da aplicação será mais fácil e rápido serem realizados.

Uma das maiores vantagens de criar uma aplicação em Java é seu grande acervo de bibliotecas gratuitas, para poder realizar trabalhos dentro da aplicação utilizando recursos não supridos pela linguagem (tais como: gráficos, relatórios, sistemas de busca, manipulação de XML, geração de código de barras, *players* de vídeo, impressão, persistência transparente, etc.)

A plataforma Java é dividida em três partes:

- JVM = ***Java Machine Virtual***, é a máquina virtual do Java que possibilita que seus códigos sejam executados em diversos tipos de dispositivos, independente do *hardware* e do sistema operacional.
- JRE = ***Java Runtime Environment***, é o ambiente de execução do Java, formado pela JVM e bibliotecas, que possibilita rodar aplicações Java.
- JDK = ***Java Development Kit***, é o pacote de desenvolvimento Java, que possibilita desenvolver aplicações Java, formado pela JRE e ferramentas de desenvolvimento, como o compilador.

3.2. SPRING

Segundo Efraim Gentil, criado por Rod Johnson em meados de 2002, o Spring é um framework de código aberto (*open source*) que possui o intuito de simplificar a programação Java, possibilitando criar aplicações que só eram possíveis utilizando EJB.

Atualmente o *Spring* possui diversos módulos o *Spring Security* (trata da segurança da aplicação), o *Spring Data* (trata da persistência no banco de dados) entre outros módulos. Mas o principal (*core*) pode ser utilizado em qualquer aplicação Java, as principais funcionalidades são a programação orientada a aspecto (AOP) e a injeção de dependência (CDI), o programador que diz ao *Spring* o que quer usar. O que faz dele uma ferramenta poderosa, pois não existe a necessidade de se usar todas as ferramentas do *framework*

para criar uma aplicação simples, a figura 4 ilustra a arquitetura do *framework Spring*.

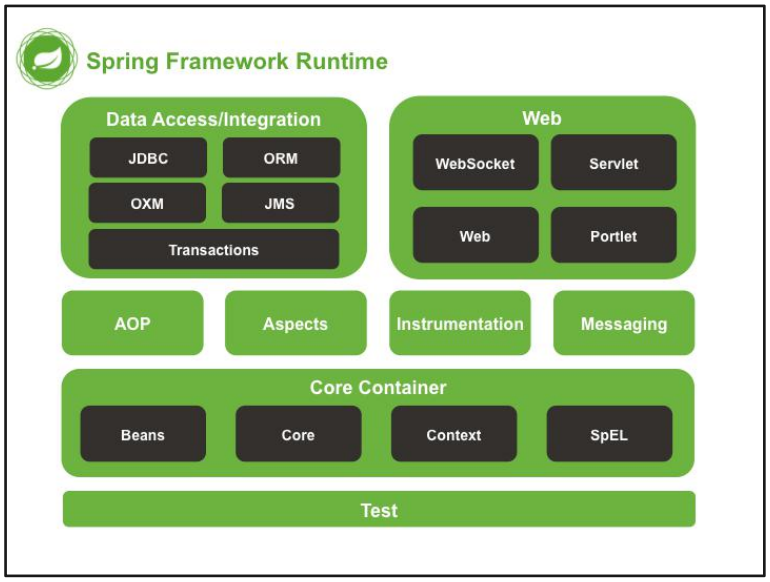

Figura 4 – Arquitetura do *Framework Spring*.

3.3. JPA

O JPA (*Java Persistence API*) veio da necessidade de melhorar a persistência dos dados no banco, pois o JDBC utiliza-se de *queries* em SQL e isso é muito gasto para o desenvolvedor na codificação sendo um problema de produtividade, e apesar do SQL ter um padrão ANSI, pode ser interpretado de forma diferente dependendo do fabricante (Higor Medeiros).

Ele é um framework leve, baseado em POJOS (*Plain Old Java Objects*) para persistir os objetos Java. Além de ser uma camada que descreve uma interface comum para *frameworks* ORM (Mapeamento Objeto-Relacional), também oferece para qualquer aplicação corporativa diversas funcionalidades essenciais. Atualmente todas aplicações de grande porte utilizam o JPA para a persistência de objetos.

3.4. BOOTSTRAP

O Bootstrap é um *framework* que auxilia o desenvolvimento de layouts responsivos para aplicações web. Criado por *@mdo* e *@fat*, dois

desenvolvedores fascinados por CSS, decidiram criar a ferramenta para otimizar a produção de *layouts* responsivos para web, o projeto era utilizado internamente no *Twitter*, mas como era *open source* e o projeto estava disponível no *GitHub*, tornou-se muito popular o uso da ferramenta tanto que hoje em dia é um requisito básico para qualquer desenvolvedor web saber usar o Bootstrap (Leandro Richard).

A maior vantagem de utilizar o Bootstrap é a capacidade dele de transformar uma página web estática em responsiva, onde os componentes presentes no HTML se adequam ao tamanho da tela do disponível que está sendo executado, porém ele não faz isso sozinho, é preciso aplicar classes nos componentes que indicam o tamanho do componente e onde ele vai ficar dependendo do tamanho da tela. A importância da responsividade é grande, pois com cada vez mais dispositivos disponíveis e o uso continuo deles pela sociedade, as pessoas estão deixando de usar somente o computador para visitar páginas na internet, a figura 5 ilustra a forma que uma página que utiliza Bootstrap se adapta ao tamanho do dispositivo que esta sendo apresentada, tornando-se responsiva.

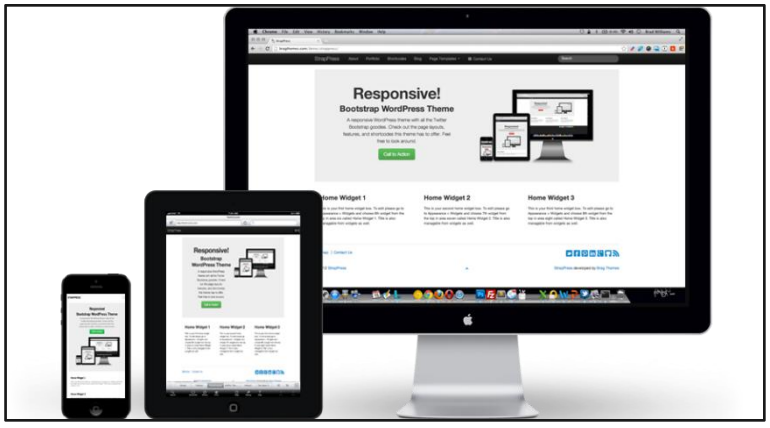

Figura 5 – Responsividade de uma página em Bootstrap em diversos dispositivos.

Hoje o Bootstrap está prestes a lançar sua versão 4, e a cada atualização é disponibilizado novos componentes, customização destes componentes e classes que auxiliam na montagem da página HTML, para que fique mais bonita, rápida e mais atraente aos olhos dos usuários.

Neste capítulo é possível concluir que o Java é uma das tecnologias mais utilizados na área do desenvolvimento, e que possui grandes e populares

tecnologias que auxiliam o desenvolver a criar aplicações melhores e com muitos recursos.

4. REST

Neste capítulo será apresentado o REST, retratando seu surgimento, vantagens do uso da tecnologia, tipo de comunicação, representações para a comunicação, métodos disponíveis e uma breve comparação entre REST e SOAP.

Criado em 2000, por Roy Fielding a partir de sua tese de Ph.D, com o intuito principal de formalizar um conjunto de melhores práticas denominadas *constraints*. Elas têm como objetivo denominar uma forma padrão de modelar como o HTTP e URI, aproveitando ao todos os recursos disponibilizados pelos mesmos.

A diferença entre os termos REST e RESTful é o que eles representam, REST é o termo que representa o conceito abstrato da metodologia de criação de uma aplicação ou API, já o termo RESTful representa a aplicação já com a implementação dos conceitos de REST e suas boas práticas.

4.1. CLIENTE-SERVIDOR

A principal característica desta *constraint* é desacoplar (separar) as responsabilidades do cliente e do servidor, chamados de *front-end* e o *back-end*. Sendo o *front-end* a parte onde o cliente interage com o sistema, sendo ela em páginas da internet em sistemas Web, ou em programas com interface em sistemas *Desktop*. E o *back-end* é a parte onde são realizadas as operações internas do sistema como regras de negócio, persistência em banco de dados, cálculos e etc.

4.2. STATELESS

Esta característica propõe que cada requisição ao servidor deve carregar todas as informações necessárias para o sucesso no tratamento pelo servidor, visto que cada nova requisição não tenha ligação com outras requisições passadas

ou futuras. O uso de *cookies* é comum neste tipo de abordagem servindo para guardar sessões no lado do servidor, mas o uso de *cookies* pode causar situações problemáticas por isso deve ser usado com muito cuidado.

Segundo o criador Roy Fielding, o uso da característica *stateless* traz a aplicação propriedades como confiabilidade, escalabilidade e visibilidade.

4.3. CACHE

Para uma melhor performance e diminuição de requisições ao servidor um sistema REST deve permitir que suas respostas sejam suscetíveis a cache. Sendo possível que uma resposta do servidor seja guardada em cache, assim uma mesma requisição que aguarda a mesma resposta não precise ser consultada ao servidor e sim ao cache, que responderá muito mais rápido ao sistema, sem a necessidade de o servidor processar a requisição novamente, a figura 6 ilustra o funcionamento das requisições enviadas do cliente para o servidor, caso o cliente possua *cache* ou não.

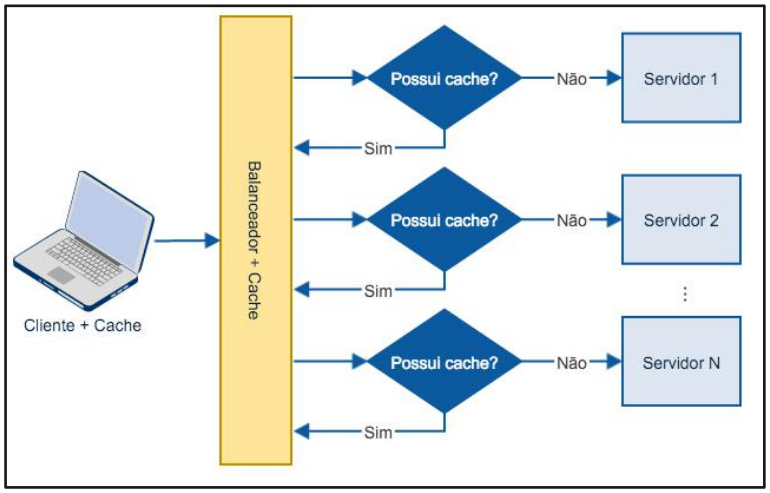

Figura 6 – Diagrama que indica uma requisição do cliente ao servidor.

4.4. SISTEMA EM CAMADAS

Com o intuito de um sistema REST ter a propriedade de escalabilidade em grandes sistemas distribuídos, ele deve ter a capacidade de adicionar elementos intermediários e que sejam transparentes para os seus clientes.

Como tratado anteriormente sobre *stateless* e *cache* além do cliente possui um ator chamado balanceador ele é um destes elementos, com ele é possível adicionarmos mais servidores para uma aplicação, sem que o cliente note sua presença.

4.5. CÓDIGO SOB DEMANDA

Quando se fala em alterações em um sistema isso pode preocupar o Analista de Sistema do software, pois se a arquitetura do sistema não foi bem planejada, onde os desenvolvedores vão programando sem rumo, qualquer alteração no sistema pode prejudicar ou até deixar de funcionar alguns módulos relacionados.

Essa característica propõe a extensibilidade que a capacidade do código de um sistema possui de se desenvolver sem ter a necessidade de quebrar outros módulos do mesmo. Possibilita adaptar o cliente com novos requisitos e funcionalidades.

4.6. RESTFUL COM HTTP

O protocolo HTTP é um dos métodos de implementação mais utilizados em aplicações REST, abaixo será descrito os principais conceitos e características de uma implementação de *WEB SERVICE* RESTful.

4.6.1. Recursos

Para identificar um objeto abstrato ou físico de forma única é utilizado um recurso, todas as características que uma URI deve seguir estão descritas detalhadamente na RFC 3986.

A modelagem de um recurso é sob um substantivo para ser representado, ao contrário das comuns API que se utiliza de verbos, como nos exemplos abaixo:

- /cliente/1
- /produto/1
- /cliente/1/notificação

Uma URI deve ser interpretada como um recurso e não como uma ação, que seria executada pelo servidor. Mas alguns autores defendem a utilização de verbos em alguns casos, porém, isso não é totalmente aceito e deve-se usar com cuidado.

4.6.2. Representações

As representações podem ser modeladas de formatos diferentes, como XML, JSON, HTML e etc., mas o que deve ser levado em consideração é que o formato escolhido deve suportar a utilização de *hypermedia*.

Essa diversa representação é graças a representação múltipla que o protocolo HTTP possui (mais conhecida como negociação de conteúdo). Para que isso seja possível deve-se utilizar o *header* "*Accept*" para especificar qual é o tipo de representação mais adequada, a figura 7 ilustra diversas representações aceitas pelo protocolo HTTP.

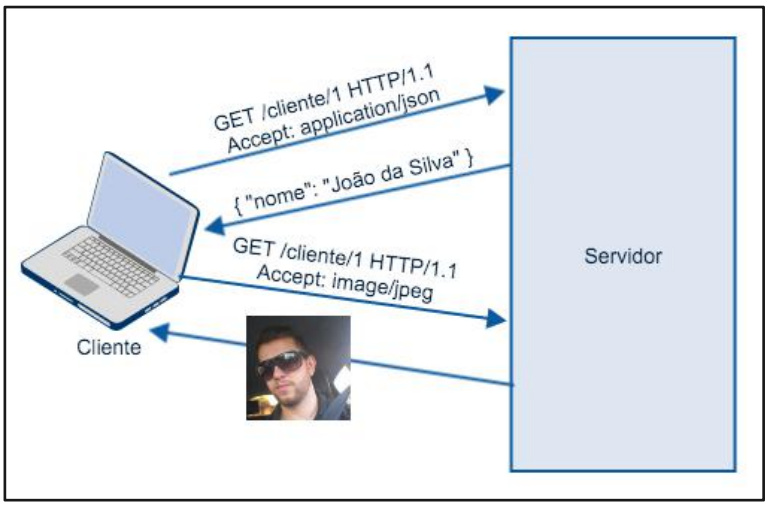

Figura 7 – Ilustração das diversas representações aceitas pelo protocolo HTTP.

Assim cada tipo de formato requisitado, o servidor retorna uma resposta mais adequada para a representação.

4.6.2.1. XML

O XML (*eXtended Markup Language*) é uma linguagem de marcação, que tem como objetivo a descrição de qualquer tipo de dados. É utilizado com formato padrão de troca de formação entre sistemas de armazenamento das mesmas. Abaixo podemos ver uma estrutura de um documento XML:

```
<cliente>
        <id>10</id>
        <nome>Alan Turing</nome>
        <nascimento>23/06/1912</nascimento>
        <profissao>Matemático</profissao>
        <endereco>
                <cidade>Manchester</cidade>
                <pais>Inglaterra</pais>
        </endereco>
</cliente>
```

Parecido com objetos do Java, as marcações do XML são escritas no formato hierárquico, mas um dos problemas encontrados no XML é a verbosidade, é preciso adicionar várias *tags* que as vezes geram um *overhead* desnecessário.

4.6.2.2. JSON

O JSON (*JavaScript Object Notation*) foi criado para melhorar a representação dos dados de forma mais simples e leve. Ele foi criado baseado na linguagem de *JavaScript* mas pode ser usado em qualquer tecnologia. Abaixo um exemplo da estrutura de um JSON:

```
{
        "id": 10,
        "nome": "Alan Turing",
        "nascimento": "23/06/1912",
        "profissao": "Matemático",
        "endereco": {
                "cidade": "Manchester",
                "pais": "Inglaterra"
        }
}
```

Os dados representados no neste JSON são os mesmos representados no XML mostrado anteriormente, porém muito mais simples de representar e interpretar.

Dentre as propriedades que a linguagem JSON possui:

- Leitura simplificada;
- Analisador simples;
- Suporte de vários *frameworks* atuais (principalmente *framework JavaScript*);
- Tamanho reduzido.

Por meio destas e outras características o JSON está cada vez mais popular na área de tecnologia para transferência de informações.

Resumindo, tanto o XML quanto o JSON, são formatos para representar os dados das aplicações, cabe a análise do cenário optar pela que melhor se encaixa de forma mais adequada. Principalmente pela simplicidade o JSON vem ganhado maior popularidade no mercado.

4.6.3. Métodos

Em uma API RESTful existem 8 métodos que podem ser utilizados de acordo com a RFC 7231, este conjunto de métodos possui a semântica de operações possíveis de serem efetuadas sob um determinado recurso. Abaixo seguirá uma descrição dos 4 mais conhecidos métodos.

4.6.3.1. GET

O método GET é utilizado quando existe a necessidade de se obter um recurso. Ele é considerado um idempotente, ou seja, independentemente da quantidade de vezes que é executado sob um recurso, o resultado sempre será o mesmo. Exemplo:

```
GET /cliente/1 HTTP/1.1
```

Esta chamada ira retornar uma representação para o recurso “/cliente/1”.

4.6.3.2. POST

O método POST é utilizado para a criação de um recurso a partir do uso de uma representação. Exemplo:

```
POST /cliente HTTP/1.1
```

4.6.3.3. PUT

O método PUT é utilizado como forma de atualização de um determinado recurso. Em alguns casos específicos ele também pode ser usado como forma de criação, como quando o cliente tem a liberdade de decidir a URI que será utilizada.

4.6.3.4. DELETE

O método DELETE é utilizado para remover um determinado recurso. Exemplo:

```
DELETE /cliente/1 HTTP/1.1
```

Além do servidor executar as requisições, ele também deve retornar uma resposta adequada para cada situação. Abaixo uma lista dos tipos de retorno suportados pelo HTTP:

- 1xx – Informações;
- 2xx – Sucesso;
- 3xx – Redirecionamento;
- 4xx – Erro no cliente;
- 5xx – Erro no servidor.

Para cada um destes tipos listados acima, existe um código especifico que pode ser aplicado em cada uma das situações encontradas na manipulação de recursos.

O HTTP também possui suporte a diversos outros recursos que podem ser muito úteis na modelagem de uma API RESTful. Entre eles pode-se citar *web linking*, negociação de conteúdo, *queries*, *chaching*, requisições condicionais e segurança.

4.7. REST VERSUS SOAP

REST como já mostrado anteriormente é um modelo arquitetural usado no projeto de aplicações da Web que contam com recursos nomeados (URL, URI, URN) e engenhosamente utiliza mais profundamente o protocolo HTTP, seu cabeçalho, seus principais métodos (GET, POST, PUT, DELETE) e toda a infraestrutura web já bem estabelecida, reconhecida e utilizada por todos. A seguir algumas das vantagens de utilizar REST:

– ~ 80% das integrações utilizam o protocolo HTTP.

– Tem o potencial de ser bem mais simples que uma implementação com WSDL/SOAP

– É mais elegante, pois utiliza ao máximo o protocolo HTTP, evitando a construção de protocolos adicionais

– A possibilidade de ter diversas representações de um mesmo recurso, por exemplo, uma dada entidade pode ser representada em diferentes formatos como JSON, XML, HTML e *text/plain*, dependendo da requisição feita pelo cliente (*Content-Negotiation*)

– Tende a ser mais performático

– Possibilidade de navegar entre relacionamentos (*Links web*) de vários recursos de forma dinâmica. Seguindo a usabilidade de qualquer sistema web. HATEOAS (*Hypermedia as The Engine of Application State*).

SOAP (*Simple Object Access Protocol*) é um protocolo para troca de informações estruturadas geralmente em uma plataforma descentralizada e distribuída. Ele se baseia em XML para seu formato de mensagem, ou seja, uma mensagem SOAP encapsula o conteúdo e pode ser trafegada via HTTP, JMS ou outro protocolo.

– É um padrão que está muito maduro no mercado, qualquer ferramenta de integração e *Framework* tem várias funcionalidades para manipular as mensagens que seguem este padrão.

– Uma mensagem SOAP pode ser propagada por diferentes protocolos, o que flexibiliza bastante várias integrações.

– É um padrão que combinado a as especificações WS-* podem garantir questões de QoS (*Quality of Service*), Segurança, transação e outras questões presentes em integrações mais complexas.

Neste capítulo é possível concluir que a tecnologia REST é robusta e possui grande potencial de expansão e de fácil domínio, possibilitando que cada vez mais desenvolvedores optem em utilizar *WEB SERVICE, para a criação de aplicações proporcionando um desacoplamento de módulos do sistema e trazendo também confiabilidade, escalabilidade, estabilidade e interoperabilidade.*

5. PROPOSTA DO TRABALHO

A proposta do trabalho é desenvolver um sistema lançador, onde a partir dele, o usuário possa ter acesso a outras aplicações, como um centralizador de autenticação.

O usuário pode ser redirecionado para as aplicações sem a necessidade de uma nova autenticação. O sistema conterá um mecanismo de permissões onde o administrador irá controlar os tipos de acesso de determinados usuários.

6. DIAGRAMAS E CASOS DE USO

Neste capítulo será apresentados os diagramas e os casos de uso.

6.1. DIAGRAMA ENTIDADE RELACIONAMENTO

A figura 8 ilustra o diagrama de entidade relacionamento, de modo demonstrar as tabelas utilizadas no trabalho.

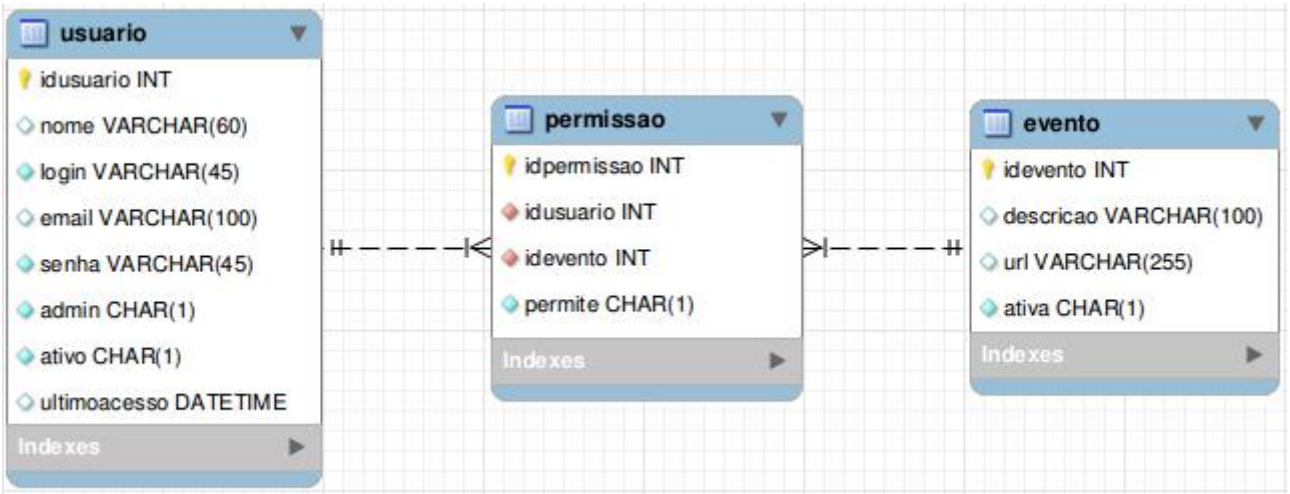

Figura 8 – Diagrama Entidade Relacionamento.

6.2. CASO DE USO GERAL

A figura 9 ilustra o caso de uso geral, demonstrando a comunicação do sistema lançador tendo o servidor REST como intermediário para ter acesso as informações do banco de dados.

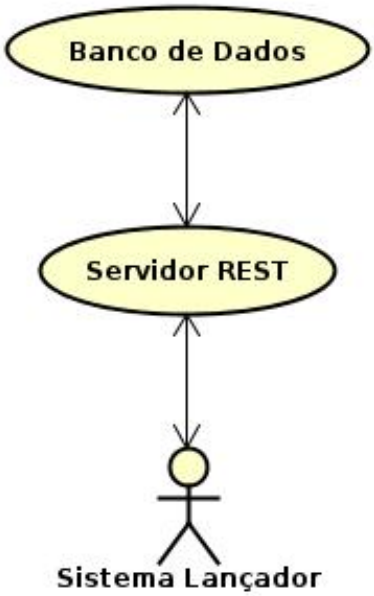

Figura 9 – Caso de Uso Geral.

6.3. CASO DE USO ESPECÍFICO

A figura 10 ilustra o caso de uso específico, demonstrando as ações que o Usuário e o Administrador têm sobre o sistema lançador.

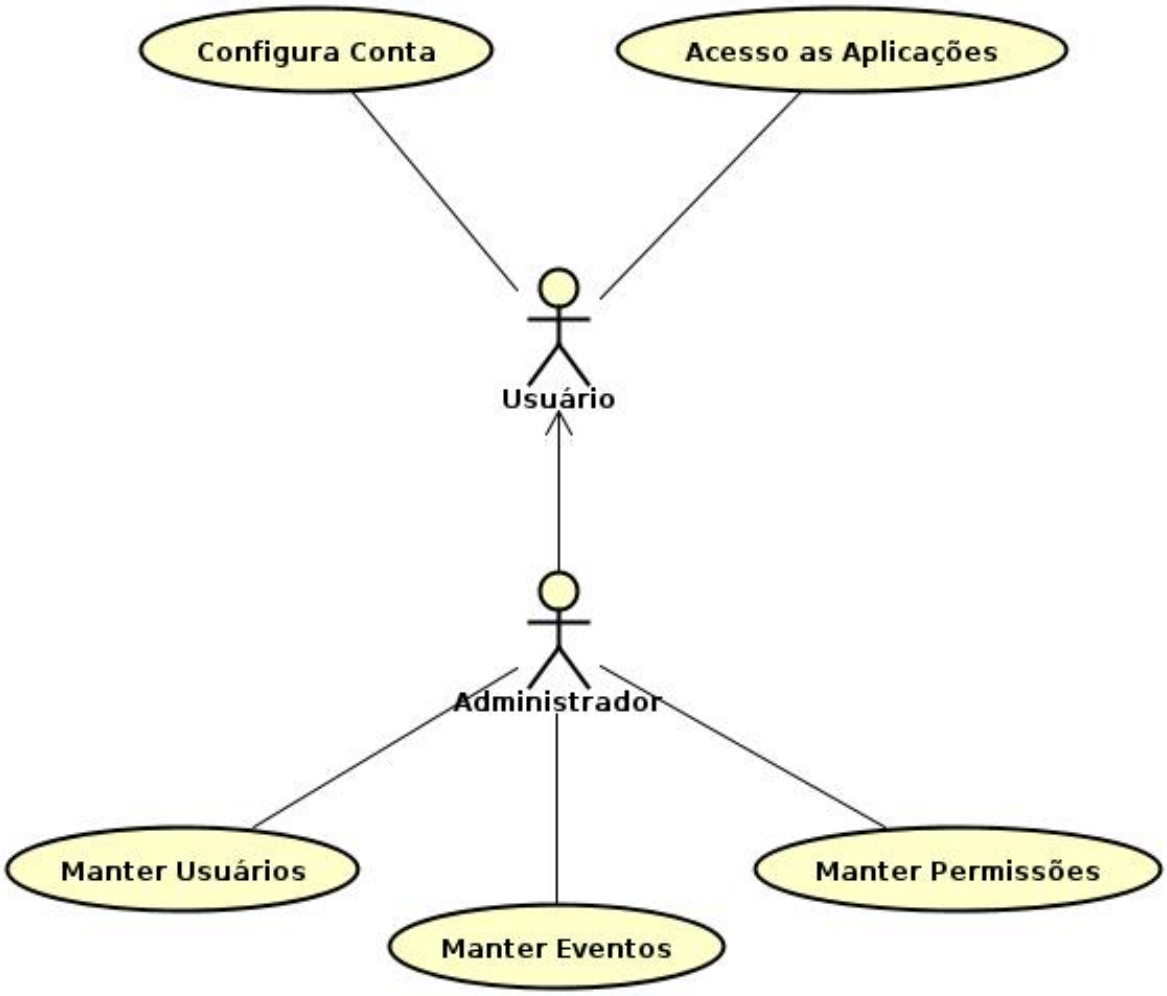

Figura 10 – Caso de Uso Específico.

6.4. DIAGRAMA DE CLASSES

A figura 11 ilustra o diagrama de classes do sistema, demonstrando as classes do sistema e a relação entre elas.

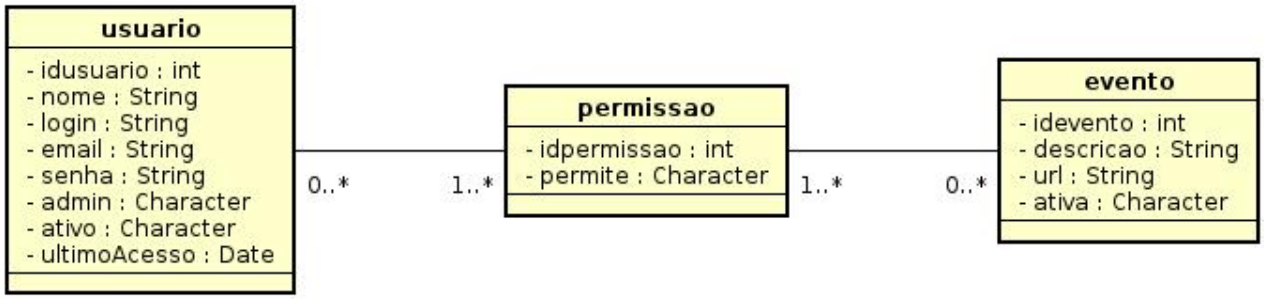

Figura 11 – Diagrama de Classes.

7. CONCLUSÃO

Com o estudo realizado neste projeto, pude concluir que a arquitetura REST é uma metodologia de *WEB SERVICE* muito utilizada por desenvolvedores, na criação de aplicações distribuídas, multiplataforma e em sistemas de integração.

O REST se popularizou por suas características de construção e pela sua representação inovadora em formato JSON que é mais simples, leve e menos verboso que o XML, sendo de fácil interpretação tanto para nós humanos quanto para as máquinas, proporcionando uma rápida transferência de informações.

Alguns trabalhos futuros que podem ser implementados como aprimorar para um sistema de *microservices*, implementar “Manter Conectado” no *login* e desenvolver o *front-end* em *FullStack Javascript.*

Com o intuito de contribuir com a comunidade de software livre, o sistema desenvolvido no trabalho esta disponível no repositório público *GitHub* e pode ser acessado através dos links :

- https://github.com/paulovandrade/lancador-rest
- https://github.com/paulovandrade/lancador-web

REFERÊNCIAS

Caelum. **O que é Java**. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/. Acesso em: 15/01/2016.

Caelum. **Uma introdução prática ao JPA com Hibernate**. Disponível em: https://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate/. Acessado em: 24/01/2016.

DIAS, Emílio. **Desmistificando REST com Java**, 1, São Paulo, AlgaWorks Softwares,11/02/2016. 42 p.

ERL, Thomas. **Introdução às tecnologias Web Services: SOA, SOAP, WSDL e UDDI - Parte 1**. Disponível em: http://www.devmedia.com.br/introducao-as-tecnologias-web-services-soa-soap-wsdl-e-uddi-parte1/2873. Acesso em: 20/03/2016.

GENTIL, Efraim (Brasil). **Introdução ao Spring Framework**. Disponível em: http://www.devmedia.com.br/introducao-ao-spring-framework/26212. Acessado em: 20/01/2016.

LANHELLAS, Ronaldo (Brasil). **CRUD completo com Hibernate e JPA**. Disponível em: http://www.devmedia.com.br/crud-completo-com-hibernate-e-jpa/32711. Acessado em: 26/01/2016.

MEDEIROS, Higor (Brasil). **Introdução à JPA – Java Persistence API**. Disponível em: http://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173. Acessado em: 26/01/2016.

RECKZIEGEL, Mauricio (Brasil). **Entendendo os Web Services**. 2006. Disponível em: http://imasters.com.br/artigo/4245/web-services/entendendo-os-webservices/. Acesso em: 14/10/2015.

REDAÇÃO iMASTERS (Brasil). **Amazon Web Services anuncia novidades para desenvolvedores móveis**. 2014. Disponível em: http://imasters.com.br/noticia/amazon-web-services-anuncia-novidades-para-desenvolvedores-moveis/. Acesso em: 17/10/2015.

RICHARD, Leandro (Brasil). **Introdução ao Twitter Bootstrap 3**. Disponível em: http://http://www.devmedia.com.br/introducao-ao-twiter-bootstrap-3/29859. Acessado em: 22/01/2016.

SAUDATE, Alexandre. **Rest Construa Apis Inteligentes de Maneira Simples**, 1, São Paulo, Casa do Código,2013. 101 p.

Wikipédia. **Ataque man-in-the-middle**. 2015. Disponível em: https://pt.wikipedia.org/wiki/Ataque_man-in-the-middle. Acesso em: 20/10/2015.