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

MARCIO ALEXANDRE DA SILVA JUNIOR

MODELO DE ARQUITETURA REST PARA USO EM WEB SERVICES E IMPLEMENTAÇÃO DE UM CHAT

Assis
2015

MARCIO ALEXANDRE DA SILVA JUNIOR

MODELO DE ARQUITETURA REST PARA USO EM WEB SERVICES E IMPLEMENTAÇÃO DE UM CHAT

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso Superior de Bacharelado em Ciência da Computação

Orientador: Me. Douglas Sanches da Cunha

Área de concentração: Ciências da Computação

Assis
2015

FICHA CATALOGRÁFICA

Da Silva Junior, Marcio Alexandre

Modelo de arquitetura REST para uso em WEB SERVICES e implementação de um chat

/ Marcio Alexandre da Silva Junior. Fundação Educacional do Município de Assis -- Assis, 2015.

68 p.

Orientador: Me. Douglas Sanches da Cunha

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1. REST. 2. Web Services 3.JAVA

CDD: 001.6

Biblioteca da FEMA

MODELO DE ARQUITETURA REST PARA USO EM WEB SERVICES E IMPLEMENTAÇÃO DE UM CHAT

MARCIO ALEXANDRE DA SILVA JUNIOR

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

Orientador: Me. Douglas Sanches da Cunha

Analisador: Prof. Esp. Célio Desiró

DEDICATÓRIA

Dedico este trabalho a todas as pessoas que fizeram parte da minha vida, amigos, familiares, professores.

AGRADECIMENTOS

Ao professor Douglas por auxiliar durante a construção deste projeto.

Aos amigos que sempre estiveram ajudando e motivando.

Aos familiares que sempre me ajudaram, motivaram e apoiaram em todos os momentos, diretamente ou indiretamente.

RESUMO

Esta monografia tem por objetivo de pesquisar e investigar o modelo arquitetural *REST* para uma implementação utilizando *Web Services*, apresentar suas características e funcionamento. Neste trabalho foram implementados dois softwares, são eles: Sistema *Client/Server* usando *REST* no servidor, para responder as solicitações de um sistema desktop de *CHAT* feito em Java. Para demonstrar o funcionamento do *REST* no *Web Services* receber requisições, processa-las e responder ao cliente por meio de *URL'S*, já o Chat é uma aplicação de bate-papo que enviam, recebem e gerenciam os grupos usando *URL'S* de requisições para o *REST*. Foram utilizadas outras tecnologias como *HTTP*, *XML* na implementação desta monografia.

Palavras-chaves: REST; WEB SERVICE; JAVA; CHAT.

ABSTRACT

This paper aims to research and investigate the REST architectural model to an implementation using Web Services, presenting their characteristics and performances. In this monograph were implemented two softwares, they are: System Client / Server using REST server, to answer the requests of a CHAT desktop system written in Java. To demonstrate the operation of REST in the web services receive requests, process them and respond to the client via URL'S, since the Chat is a chat application that send, receive and manage the groups using URL'S requests for REST. Other technologies such as HTTP, XML were used in the implementation of this paper.

Keywords: REST; WEB SERVICE; JAVA; CHAT.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HTML - *Hypertext Markup Language*

XML – *eXtensible Markup Language*

REST- *Representational State Transfer*

URI – *Uniform Resource Identifier*

API – *Application Programming Interface*

URL – *Uniform Resource Locator*

IBGE – Instituto Brasileiro de Geografia e Estatística

HTTP – *Hypertext Transfer Protocol*

SSL – *Security Socket Layer*

SOAP – *Simple Object Access Protocol*

W3C – *World Wide Web Consortium*

MEP – *Message Exchange Pattern*

JKS – *Java Key Store*

SGML – *Standard Generalized Markup Language*

DTD – Definição de Tipos de Dados

JDK – *Java Development Kit*

IDE – *Integrated Development Environment*

SUMÁRIO

1 INTRODUÇÃO

A Internet faz parte das tecnologias que mais crescem no mundo, utilizada em vários tipos de aparelhos e dispositivos, como computadores pessoais (Desktops,Notebooks,Netbooks), celulares, *tablets*, relógios, televisões dentre outros. Por encontrar-se em vários aparelhos e alguns deles serem móveis, faz com que a internet seja de fácil aceso, utilizada em qualquer lugar, com isso sua utilização aumenta.

Algumas atividades, especificamente os serviços web, utilizam uma tecnologia chamada de *Web Services*. A *Web Services* é uma solução para fazer a integração de sistemas, ou seja, fazer com que novos sistemas possam interagir/conversar entre si. Também é importante ressaltar que esta tecnologia não depende da linguagem na qual o sistema foi criado, pois ela é traduzida para um padrão XML (*eXtensible Markup Language*), portanto é universal para web. (TITTEL, 2002)

Em *Web Services,* existe uma tecnologia conhecida como *REST* (*Representational State Transfer* – Transferência do Estado Representativo) que foi desenvolvida por Roy Fielding em seu doutorado, nos meados do ano 2000. Esta tecnologia tem como alvo todo e qualquer serviço Web e que utiliza a Web Services, “Em REST, tudo é definido em termos de recurso, sendo estes os conjuntos de dados que são trafegados pelo protocolo. Os recursos são representados por *URI’s* ” (*Uniform Resource Identifier*). (SAUDATE, 2014b)

Grandes empresas utilizam a arquitetura *REST* para melhorar seus serviços, e deixá-los para que usuários possam utilizar dos serviços sem que acessem diretamente o servidor. O *Google*, por exemplo, disponibiliza uma *API*, para utilização do *Google Maps*. Esta *API* contém alguns métodos prontos na qual devem ser passados por parâmetros, como longitude e a latitude. O método conecta-se ao *Web Services* e retorna para a aplicação os resultados desta requisição, que neste caso será nome da cidade que pertence os dados solicitados através do parâmetro.

Um dos problemas desta área é a dificuldade na integração de sistemas (*softwares*) novos com os antigos. Há casos, em que um sistema não foi projetado para interagir com outro, levando a empresa a ter que repensar seu software para que possa ser reaproveitado por outros, podendo causar atrasos na implementação de novos sistemas, ou até mesmo ter que atualizar os sistemas antigos, para atender uma necessidade de integração.

1.1 OBJETIVOS

O objetivo desta monografia é pesquisar a arquitetura *REST*, assim como todas as tecnologias que envolvem o desenvolvimento de um sistema Cliente/Servidor completo. E a implementação de uma aplicação utilizando estas tecnologias *REST*, com Java para que possa demostrar a facilidade na integração dos sistemas, como um middleware, recebendo uma requisição e devolvendo o objeto pronto para uso.

1.2 JUSTIFICATIVAS

A proposta deste trabalho tem como principal ideia, demonstrar a facilidade na integração entre sistemas, fazer com que o código fonte dos sistemas possa ser reaproveitado e com isso evite com que ações sejam reescritas.

1.3 MOTIVAÇÕES

A grande motivação para entrar nesta área é seu crescimento, facilidade e distribuição, área de segurança, funcionalidade com qualquer banco de dados, podendo usar até vários ao mesmo tempo. Muitas empresas já aderiram ao *REST*, *Google*, *Facebook*, *Twitter*, Terra, dentre outras. Esta tecnologia facilita muito a utilização de requisições de um servidor, onde as empresas decidem se a requisição pode ser realizada ou recusada por meio do *REST*, pois, algumas utilizam *API's* outras aceitam uma requisição diretamente pela *URL*.

1.4 PERSPECTIVAS DE CONTRIBUIÇÃO

Este trabalho tem como proposta de contribuição, ajudar as pessoas entender onde e quando é viável a utilização da tecnologia *REST* e suas possíveis atuações, levando em consideração Cliente/Servidor. Demonstrar seu funcionamento e sua facilidade em integração de sistemas modulares e não modulares e autenticação de usuários. Assim, elas podem decidir, como e quando será o melhor momento para utilização.

1.5 ESTRUTURA ANALITICA DO PROJETO

O capítulo 2, descreverá o levantamento bibliográfico das tecnologias utilizadas no projeto. Modelo das etapas para desenvolvimento do projeto onde será detalhado a partir do capítulo 3. O capítulo 4 descreverá a análise orientada a objeto, onde contém os diagramas e os detalhes dos eventos e ações dos sistemas.

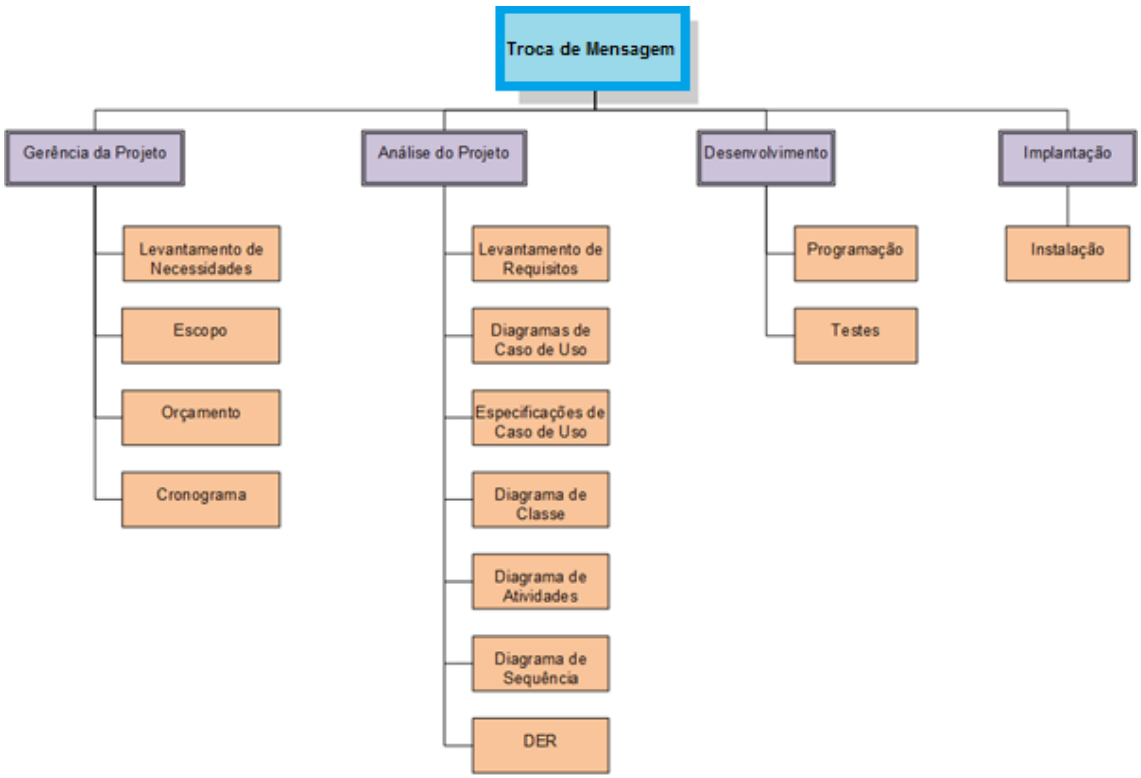

Figura 1 - Estrutura Analítica do Projeto

A Figura 1 representa o Estrutura na qual será seguida para a realização deste projeto, e será descrita a partir do capítulo 3.

2 REVISÃO BIBLIOGRÁFICA DAS TECNOLOGIAS

A seguir serão descritos alguns dos recursos, tecnologias que são importantes para o desenvolvimento de aplicações e a implementação.

2.1 INTERNET

A internet surgiu entre as décadas de cinquenta e sessenta durante a Guerra Fria, com o objetivo de unir centros militares e universidades Norte Americanas. O propósito para a criação da ARPNet (como era chamada no início de sua criação), criar uma rede na qual a comunicação não seria perdida e as informações não seriam destruídas sob um ataque da União Soviética. (DUMAS)

A internet entrou para a rotina da população, seja para acesso as redes sociais, compras, pesquisas ou trabalho, ela ganha proporções gigantes a cada ano. Dados do site *internetworldstats* (2014) mostra que em 14 anos a internet na América latina cresceu cerca de 1.571,4%. Segundo dados do IBGE (Instituto Brasileiro de Geografia e Estatística) de 2005 a 2011 no Brasil houve um crescimento de mais de 100% na utilização da internet de pessoas com mais de 10 anos.

2.2 WEB SERVICES

A *Web Services* é um serviço independente de plataforma, ou seja, pode ser acessado de um computador, celular ou televisão sem nenhuma restrição. Ele utiliza a arquitetura Cliente/Servidor, com isso o *Web Services* pode ser visto e utilizado em qualquer lugar com acesso à internet. (GURUGE, 2004)

A *Web Services* é uma aplicação web distribuída, multi plataforma, ou seja, um aplicativo pode ser executado de qualquer dispositivo. Ele é feito com base no *HTTP*, esta tecnologia independe da linguagem na qual o sistema foi criado, pois ela é traduzida para *XML*, na qual é universal para web. (KALIM, 2010)

As *Web Services* começaram a crescer e, como forma de organizar os dados destes serviços, a W3C criou a Web Semântica. Seu objetivo é fazer com que o

serviço possa buscar dados de forma mais interativa, e que facilite a busca por dados já que ela viabiliza o banco de dados. (W3C, 2014b)

2.2.1 DEFINIÇÃO

Segundo Kalin (2010) *Web Services* é uma aplicação web, ou também pode ser chamada de aplicação distribuída, pois seu objetivo, é estar em qualquer dispositivo. A *Web Services* é fornecida para seus usuários usando o protocolo padrão *HTTP* (*Hyper Text Transport Protocol*), que é responsável pelo transporte das requisições. A Web Services pode ser dividida em três características para diferencia-la dos outros tipos de *software*, são elas: *infraestrutura aberta*, *transparência de linguagem* e *design modular*.

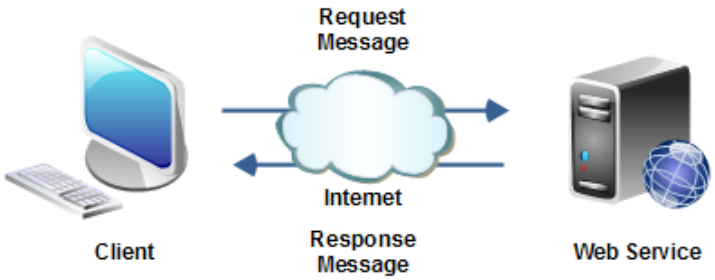

Figura 2 - Representação Básica de WebService

Kalin (2010) diz que uma *Web Services* pode ser dividida em dois grupos, *SOAP* e *REST*, porém também é possível utilizar ambos juntos.

2.2.2 VANTAGENS

A vantagem de utilizar um *Web Services* além de estar disponível em qualquer dispositivo é também que, a linguagem da aplicação cliente pode ser diferente da linguagem da aplicação servidor. Uma *Web Services* também pode ser criada de forma modular, para várias aplicações conversarem entre si. (KALIN, 2010)

2.3 SOAP

O *SOAP* (*Simple object Access Protocol*) foi criado pela *W3C* (*World Wide Web Consortium*), sua função é a mesma que um envelope, dividido entre *Header* e *Body*. Onde o *body* armazena os dados da requisição, por exemplo, os parâmetros de uma busca, e o header que contém os metadados da requisição. (SAUDATE, 2014a)

SOAP é um protocolo simples e comum, destinado a troca de informação estruturada, porém descentralizada. Suas duas maiores vantagens são: simplicidade e extensibilidade. Esta tecnologia usa o *XML* para definir a extensibilidade da mensagem. (W3C, 2007a)

O Protocolo de Acesso Simples a Objetos conhecido como *SOAP* (*Simple Object Access Protocol*) é um protocolo que encapsula a requisição como um envelope, para transporte dos dados. Ele é uma implementação a nível de programação, é uma estrutura tipo *Message Exchange Parttern* (*MEP*) solicitação/resposta, onde o cliente faz uma requisição o *SOAP* envelopa a requisição, o servidor recebe o envelope descapsula executa a requisição, envelopa a resposta e envia de volta para o cliente. (KALIN, 2010)

A figura 3 apresenta a estrutura de funcionamento do protocolo *SOAP.*

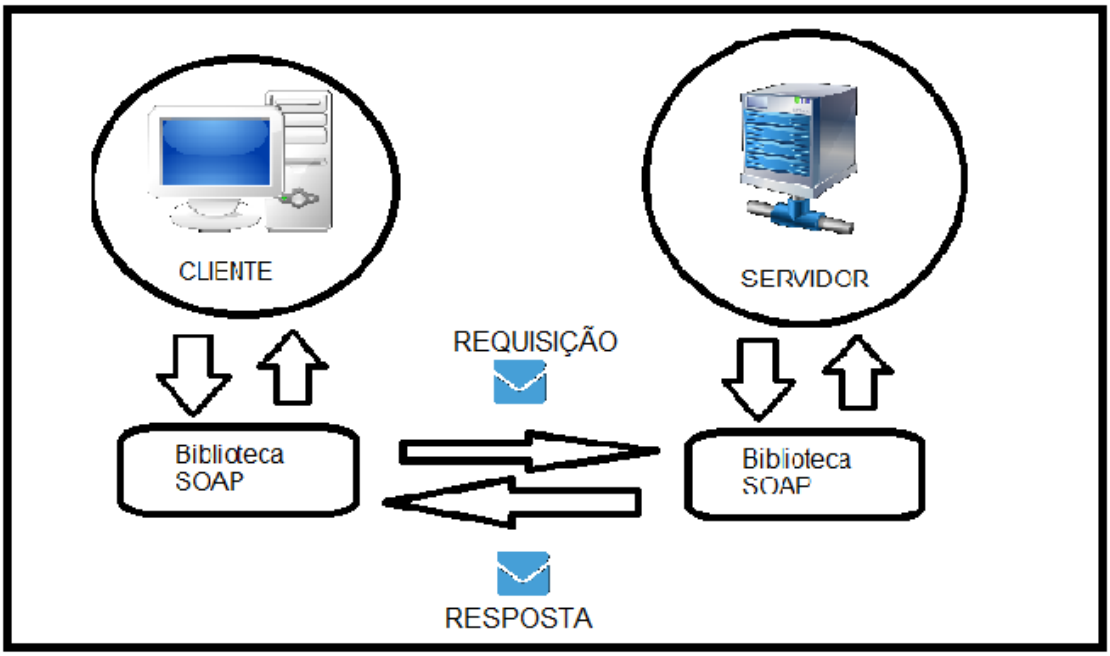

Figura 3 - Representação Protocolo SOAP

2.4 SEGURANÇA

Esta seção aborda o conceito de segurança, suas definições, principais meios e modos de proteger uma aplicação e os dados trafegados, utilizando protocolos e bibliotecas de segurança. As seguranças em Web Services são de extrema importância, pois, quando se faz uma requisição em *HTTP* ela acaba passando por vários roteadores, e não se pode confiar que estes equipamentos sejam seguros. (SAUDATE, 2013a)

A segurança em uma *Web Services* é algo realmente muito importante, pois há, troca de dados e não deve correr o risco de ser interceptado. A segurança básica em uma *Web Services* a nível de programação são: segurança de baixo nível, autenticação de usuário e uso de protocolos de segurança. (KALIN, 2010)

Ter a certeza que a mensagem chegará ao destinatário correto, codificar a mensagem para que um intruso não consiga compreender os dados caso seja interceptada. (KALIN, 2010)

2.4.1 DEFINIÇÃO

Segundo o dicionário Aurélio, “segurança” pode ser definido como um estado, qualidade ou condição de seguro, e a palavra “seguro” tem como significado: ser livre de perigo ou risco.

2.4.2 TIPOS DE SEGURANÇA

A principal preocupação na segurança da *Web Services* é a interceptação dos dados, pois como foi citado, os pacotes de uma requisição podem passar por roteadores ou *switches* desconhecidos. Um dos modos mais utilizados para implementação neste caso, é o uso do *HTTPS.* (SAUDATE, 2013a)

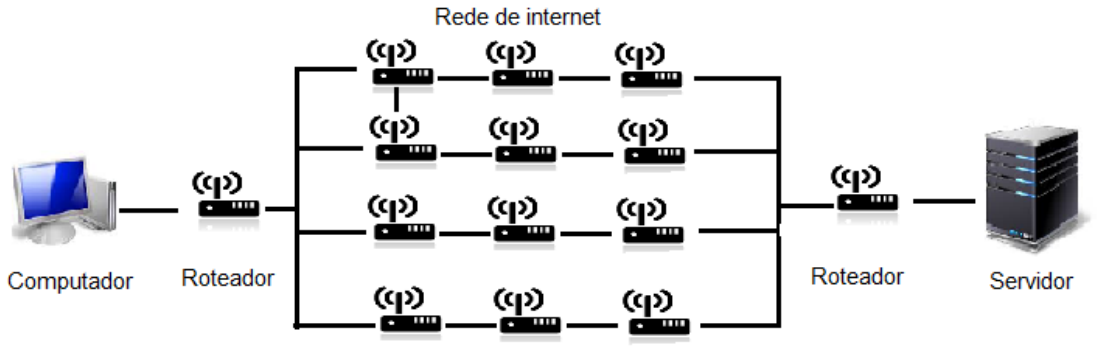

Figura 4 - Representação da estrutura de internet

2.4.2.1 *HTTPS*

O *HTTP* (*Hyper Text Transfer Protocol*) é um protocolo de transporte que utiliza-se do cabeçalho para transportar os dados. Com isso, os dados dentro da área de metadados são transportados com mais segurança, por causa do *SSL* (*Secure Sockets Layer*). Ele é um dos protocolos mais utilizados, quando necessita-se transportar dados via web. (SAUDATE, 2014a)

O *HTTPS* não é uma variação ou tipo diferente de *HTTP*, ele é somente uma camada a mais no *HTTP* que proporciona a segurança no protocolo, que é responsável pelo transporte de dados. (SAUDATE, 2014a)

O *Hipertext Transfer Protocol over Secure Socket Layer* (*HTTPS*) utiliza certificados digitais para autenticação, assim garantindo a autenticidade dos dados. O uso do *HTTPS* pode ser facilmente reconhecido quando a sigla está presente antes do endereço eletrônico, por exemplo, “https://www.google.com”.

A Linguagem de programação *JAVA* tem um sistema para este tipo de certificado, chamado de *java key store – jks*, esta ferramenta permite criar um certificado especificando vários dados, como sua validade, qual algoritmo de criptografia será usado, o domínio. (SAUDATE, 2013a)

O *HTTPS* permite escolher entre dois modos de autenticação, o *basic* e o *digest.*

2.4.2.1.1 *HTTPS BASIC*

A autenticação *basic*, é um modo simples onde é passada dentro do header do *HTTP* uma *tag* chamada de *authorization* que contém um usuário e uma senha. Estes dados são convertidos de binário para alfanuméricos utilizando o algoritmo Basic64. (SAUDATE, 2013a)

Um arquivo “*.xml”* deve ser configurado no servidor de aplicação (para o exemplo será utilizado *GlassFish*), sua localização é no *Security Manager*, assim autenticação pode ser respeitada.

A figura 5 demonstra o conteúdo do arquivo de configuração *XML*.

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Cidades</web-resource-name>
                <url-pattern>/cidades/*</url-pattern>
                <http-method>GET</http-method>
                <http-method>POST</http-method>
                <http-method>DELETE</http-method>
                <http-method>PUT</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>ADM</role-name>
        </auth-constraint>
</securaty-constraint>
```

Figura 5 - Arquivo de configuração HTTPS Basic

A *tag* “web-resource-name” serve para dar um nome aquela configuração, “*url-pattern*” é a *tag* responsável por especificar quais *url’s* terão esta configuração, o “*http-method*” diz qual os métodos serão afetados pela configuração. “*auth-constraint*” serve para dizer quais usuários poderão ter acesso aos métodos, e a “*role-name*” diz qual o usuário.

2.4.2.2 WS-SECURITY

Seu objetivo é melhorar a segurança nos *Web Services* baseados em *SOAP*.

A linguagem JAVA contém uma biblioteca de *Web Services*, chamada de jax-ws. Com ela temos vários métodos próprios do JAVA para dar mais segurança à *Web Services*, junto com o protocolo *HTTPS*. A jax-ws implementa confiabilidade e autenticidade.

2.4.2.3 CRIPTOGRAFIA

Apesar de ter várias formas de implementar segurança no *Web Services*, elas não fogem da criptografia.

Cifrar é um método que se utiliza para esconder em mensagem embaralhar os dados ou até mesmo substituir. Durante as guerras, para se comunicarem com outros postos de guerras que estavam mais distantes, os países cifravam as mensagens para dificultar a compreensão dos inimigos, caso a mensagem fosse interceptada. (TERADA, 2008)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
z	x	c	v	b	n	m	a	s	d	f	g	h	j	k	l	q	w	e	r	t	y	u	i	o	p

Figura 6 - Exemplo de criptografia I

Acima temos um método simples de cifrar, onde as letras da primeira linha serão trocadas pelas letras da segunda linha, ou seja, a palavra “COMPUTADOR” seria cifrada para “CKHLTRZVKE”. Deste modo a compreensão para quem não sabe que a palavra foi cifrada se torna mais difícil.

A criptografia tem como objetivo embaralhar a informação antes de enviar ao recebedor. Há vários modos para cifrar dados, porém ninguém conseguiu criar um algoritmo de criptografia que fosse totalmente seguro.

Os algoritmos para cifrar mais comuns utilizam chaves para decifrar ou, até mesmo chaves para passar a outro nível de cifrar/decifrar.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
z	x	c	v	b	n	m	a	s	d	f	g	h	j	k	l	q	w	e	r	t	y	u	i	o	p
q	w	e	r	t	y	u	i	o	p	a	s	d	f	g	h	j	k	l	z	x	c	v	b	n	m

Figura 7 - Exemplo de criptografia II

O exemplo acima usou a primeira linha como base, e pode escolher qual linha usar para cifrar, linha 2 ou 3. Neste caso a palavra “COMPUTADOR” ficará “EGDHXZQRGK”, para usar este método é necessário que o receptor saiba qual é a chave utilizada, neste caso a chave é a linha 3. Algumas técnicas de criptografia armazenam a chave junto com a palavra codificada.

2.5 *REST*

Nesta seção será abordado o *REST*, suas definições, vantagens, usabilidade, exemplos práticos.

2.5.1 DEFINIÇÃO

Segundo Saudate (2014), *REST* significa *Representational State Transfer*. *REST* é um modelo arquitetural concebido por um dos autores do protocolo *HTTP* chamado Roy Fielding. Sua plataforma aproveita a capacidade do protocolo em diferentes métodos de comunicação, utilização do header *HTTP*, utilização de arquivos como recursos e utilização de *media types*.

O *REST,* diferentemente do *SOAP*, não é um protocolo e sim um estilo de arquitetura para *Web Services*. Para este estilo tudo é definido como recurso, e eles são representados por *URI's.* Trabalha com Cliente/Servidor, envio de requisições e respostas por meio de parâmetros, estas requisições e respostas são trafegadas pelo protocolo *HTTP*. Os parâmetros de requisição e resposta podem ser de vários tipos, sendo os principais o *XML* e o *JSON. (SAUDATE, 2013b)*

Representation State Transfer- Transferencia de Estado Representacional (REST) segundo Kalim (2010) tem como principal foco um recurso. O recurso pode ser qualquer coisa, porém obriga a ter um *URI*, chamado também de *Hyperlink*. O *URI* é um identificador de recurso, e é utilizado para especificar qual recurso do sistema está sendo invocado.

O *REST* usa o protocolo de transporte *HTTP*, que não é somente um protocolo, mas também uma API, que contém verbos chamados de métodos. Esses métodos são usados como *CRUD* (*Create, Read, Update, Delete – Criar, Ler, Atualizar, Deletar*) são eles: *Post, Get, Put, Delete.* Apesar de serem métodos eles também são recursos que o usuário pode requisitar por meio do *URI*, estes métodos de chamada ficam dentro do *Body* do *HTML.* (KALIM, 2010)

Na tabela 1 Métodos/Verbos REST é apresentado um modelo de cadastro de cidade que apresenta o funcionamento das chamadas dos métodos no *URI*.

Métodos	Ações	URI
POST	Cria uma nova cidade	127.0.0.1:8080/Cidades/nova/nome=Assis&id=28
GET	Lê/ busca uma cidade	127.0.0.1:8080/Cidades/buscar/id=28
PUT	Atualiza uma cidade	127.0.0.1:8080/Cidades/atualizar/id=28&nome=SãoPaulo
DELETE	Deleta uma cidade	127.0.0.1:8080/Cidades/deletar/id=28

Tabela 1 - Métodos/Verbos REST

Neste exemplo o *IP* do servidor é local *Host* 127.0.0.1, em seguida a porta do servidor de aplicação que é o :8080, o próximo é o nome da aplicação *REST* que está no servidor /Cidades, em seguida uma identificação que nos diz qual método será requisitado /nova cria, /buscar lê, /atualizar atualiza e /deletar deleta. Em seguida os parâmetros. (KALIM, 2010)

As respostas destas requisições podem ocorrer de várias maneiras como, por exemplo, *media type* onde tem como função definir como será o retorno do conteúdo. O cliente pode fazer uma requisição onde se utiliza o *URI* 127.0.0.1:8080/Cidades/buscar/id=28 isto significa que deseja buscar pela cidade de ID=28, o retorno para a aplicação pode ser de vários tipos dependendo da necessidade do cliente, tendo a possibilidade de retornar uma imagem, *XML*, *JSON*, *HTML* ou texto. (SAUDATE, 2013b)

O tipo de *media type* que será retornado pode ser escolhido pelo próprio cliente durante a construção da requisição. No *HEAD* (cabeçalho HTTP) no campo

Accept:, de acordo com o valor do parâmetro passado, por exemplo "*application/xml*" *o* retornará um *XML*.(SAUDATE, 2013b)

O *REST* tem como grande vantagem a simplicidade, diferente do *SOAP* que trabalha com mensagens dentro de mensagens. Isso faz com que o processo para descapsular os dados seja mais complexo, porém mais seguro contra interceptação de pacotes por hackers. (KALIM, 2010)

2.6 XML

O *XML* é muito utilizado para transporte de dados via web, além de sua interoperabilidade, tem-se a vantagem de ser fácil entendimento, graças a sua estrutura simples e facilmente modelada. (TITTEL, 2002)

Criada em 1996 a *XML* tem como objetivo ser uma linguagem compreendida por softwares e que fosse possível usa-la para ser trafegada pela internet e que pudesse ser integrada a outra linguagem como *HTML* (*Hipertext Markup Language* - linguagem de marcação para Hipertexto) e *SGML* (*Standard Generalized Markup Language* – Linguagem de Marcação Padrão Generalizada) e legível para Humanos. (TITTEL, 2002)

Segundo Tittel (2002) o *XML* é "especial" por causa de sua flexibilidade, tornando qualquer informação que o usuário possa precisar para trafegar dados entre sistemas ou Web.

A imagem abaixo mostra uma estrutura de *XML* básico, como base será utilizada o exemplo da tabela 1.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
    <Cidades>
        <Cidade>
            <Nome>Assis</Nome>
            <Estado>São Paulo</Estado>
            <Codigo>1</Codigo>
        <Cidade>
    </Cidades>
```

Figura 8 - Estrutura básica de um arquivo xml

TAG “<?xml version = “1.0” encoding = “UTF-8”?>” padrão do *XML*, onde todos os arquivos *XML* devem tê-la. O atributo “version” refere-se à versão do *XML* que está sendo utilizada, porém só existe a versão 1.0, mesmo assim é obrigatório. O segundo atributo é o “*encoding*” que representa qual codificação de caracteres este arquivo representará.

Um *XML* precisa de uma *TAG* para indicar seu início, conhecida como “*ROOT*” (raiz) é dela que se originam os dados, na figura 7 a *TAG* “Cidades” será a Raiz. Dentro da Raiz a *TAG* de “Cidade” que representa um objeto do tipo cidade, e dentro dele têm-se os dados do objeto, como o nome, estado e código.

2.6.1 DTD’S

O *DTD* é responsável por validar o *XML*, definindo quais elementos e quais atributos eles terão. Com isso ele forma uma estrutura que o *XML* tem que seguir para ser validado.

Segundo Tittel (2002) Para a criação de um *DTD* deve-se ter conhecimento sobre sete blocos básicos: *Elementos*, *Atributos*, *Entidades*, *PCDATA*, *CDATA*, *Marcas* e *Notações*.

- Elementos: Componente principal.
- Atributos: Informações para o elemento
- Entidades: Referência a textos
- Marcas: Delimitador de elementos
- Notações: Referência à plug-ins
- PCDATA: Dados analisados
- CDATA: Dados não analisados

3 LEVANTAMENTO DE REQUISITOS

Neste capítulo será feito o levantamento de requisitos do sistema, para demonstrar o funcionamento do sistema *REST*. Junto com a aplicação em *REST* será desenvolvido um sistema *Desktop* de troca de mensagens entre grupos de usuários.

3.1 SISTEMA DESKTOP

O sistema de troca de mensagem será desenvolvido na linguagem JAVA, com isso o sistema poderá ser utilizado de qualquer computador independente de seu Sistema Operacional. Para construção da interface será utilizado a biblioteca *SWING* do Java. O banco de dados utilizado será o PostgreSQL.

O objetivo desta aplicação é um usuário administrador criar grupos e popular estes grupos com usuários. Estes usuários poderão enviar mensagens para todos participantes do grupo. A cada período de tempo o sistema irá enviar uma requisição ao sistema *REST* para receber atualizações de mensagens.

3.2 SITEMA REST

O sistema REST será desenvolvido em JAVA e funcionará como um *middleware* onde a entrada será a requisição do *software desktop*, uma verificação será feita para ter a certeza de que o cliente realmente está no grupo e pode receber ou enviar mensagem. Se estiver o *REST* acessará o banco de dados, e depois retornará ao cliente a nova mensagem ou enviará sua mensagem aos outros participantes do grupo.

3.3 ESTRUTURA REST

A figura 9 representa a estrutura de *REST* a ser seguida, onde qualquer dispositivo pode acessar por meio da uma conexão com o serviço *REST*. Chegando à Web Services, ele terá quatro camadas, segurança, o servidor

REST, serviços e o *DAO*. A segurança é responsável pela autenticação do usuário, e também por cifrar a requisição que foi cifrada no programa desktop. O servidor *REST* encaminha a requisição para o serviço determinado, no serviço é feito as verificações e regras de negócios, após estas ações o serviço vai para a camada de *DAO* para realizar a conexão com o banco.

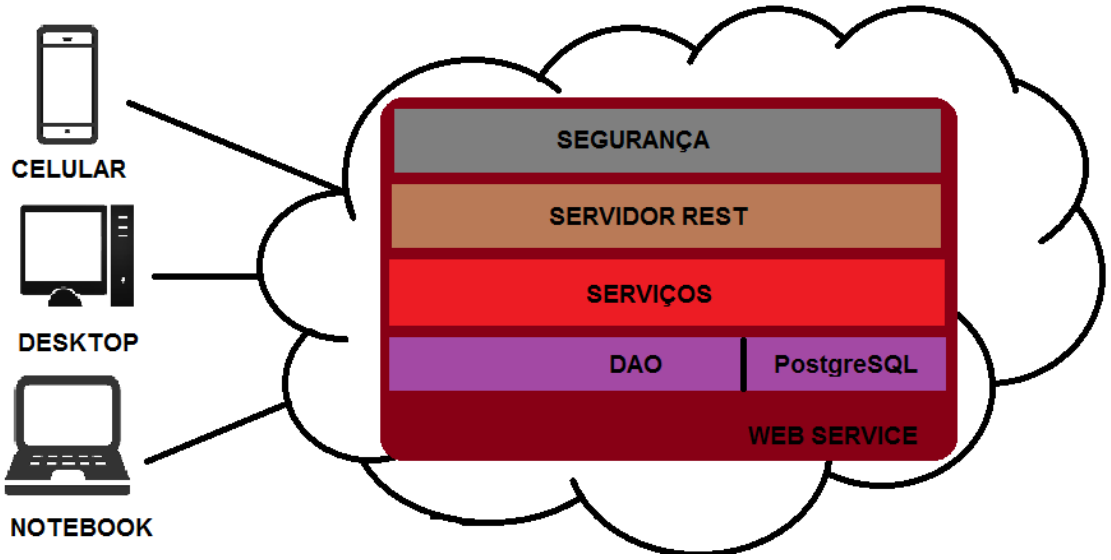

Figura 9 – Estrutura REST

3.4 DETALHAMENTO DO PROBLEMA A SER RESOLVIDO

Este projeto tem como objetivo resolver os problemas como autenticação de usuários por meio de uma *Web Services* baseado na arquitetura *REST*. Fazer integração entre sistemas, e autentica-los.

3.5 RESULTADOS ESPERADOS NA IMPLEMENTAÇÃO DO SOFTWARE

Para este projeto, espera-se que a utilização desta tecnologia venha a agilizar, facilitar e que possa reaproveitar o código para a utilização de autenticações de usuários, dentro de vários sistemas.

3.6 FORMA ADOTADA PARA LEVANTAMENTO DOS REQUISITOS

A análise para este sistema foram feitos de duas formas: *REST* e *desktop*. Onde a análise do *REST* foi feita com base na autenticação de usuário, e depois a necessidade do *software desktop*. A parte *desktop* foi feito um levantamento das necessidades de uma simples aplicação de troca de mensagens entre usuário/grupo.

3.7 RESTRIÇÕES DE DESENVOLVIMENTO DO SOFTWARE

Para o desenvolvimento deste sistema, será necessário um microcomputador, uma *IDE* (*netbeans* ou *eclipese*), banco de dados PostgreSQL, *JDK*, e acesso à internet.

Para execução do sistema, será necessário um microcomputador com acesso à internet.

3.8 PROBLEMAS POTENCIAIS

Cifrar e decifrar dados;

Transporte dos dados;

Conversão para XML;

3.9 PRIORIZAÇÃO DA IMPLANTAÇÃO DOS REQUISITOS

Serviço de autenticação;

Serviço REST (geral);

Login (desktop);

Cadastro de usuário;

Cadastro de grupo;

Envio de mensagem;

Recebimento de mensagens;

3.10 LEVANTAMENTO DE CUSTO

Para desenvolvimento deste projeto serão necessários os itens relacionados abaixo:

Item	Custo
Computador (Linux)	R$ 1.400,00
IDE NetBeans 8.0	Gratuita
Banco de Dados PostgreSQL	Gratuito
Servidor Java	R$ 50,00 por mês
Analista de Sistema Web Junior	R$ 6.200,00

Tabela 2 - Levantamento de Custo

3.11 CRONOGRAMA

Abaixo segue o cronograma seguido para a realização deste projeto.

Atividades/mês	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set
Levantamento bibliográfico												
Desenvolvimento do Texto de Qualificação												
Levantamento de requisitos												
Entrega da Qualificação												
Elaboração do projeto												
Implementação do sistema												
Testes												
Escrita do relatório final e defesa												
Entrega da Defesa												

Tabela 3 - Cronograma

4 ANÁLISE ORIENTADA A OBJETO

Neste capítulo será descrito toda a análise de requisito para sistema desktop e sistema *REST*, com os diagramas.

4.1 LISTA DE EVENTOS

Nº	**Descrição**	**Caso de Uso**
01	Administrador cadastrar usuários	Cadastro de Usuário
02	Administrador cadastrar grupos	Cadastro de Grupos
03	Administrador ativar e desativar usuário	Gerenciamento de Grupos
04	Login	Login
05	Enviar mensagem	Envio de Mensagem
06	Receber mensagem	Recebimento de Mensagem
07	Visualização de Mensagem	Visualização de Mensagem

Tabela 4 - Lista de Eventos

4.2 DIAGRAMA DE ENTIDADE E RELACIONAMENTO

Na figura abaixo temos uma representação do diagrama de entidade e relacionamento do banco de dados do projeto.

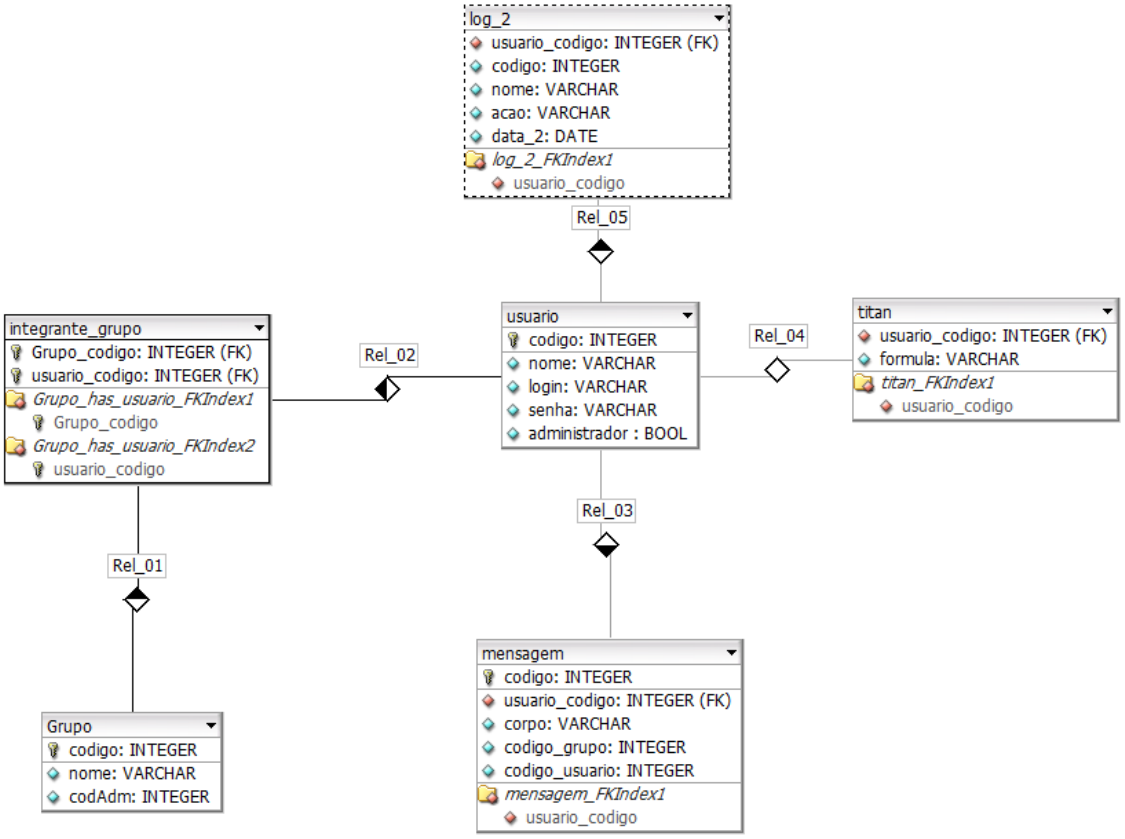

Figura 10 – Diagrama de Entidade e Relacionamento

4.3 DIAGRAMA DE CASO DE USO GERAL

Este diagrama de caso de uso demonstra as operações possíveis para cada ator. O usuário poderá realizar quatro operações, enviar mensagem, receber mensagem, fazer login e visualizar mensagem. As três primeiras operações descritas serão realizadas por meio do envio de uma requisição na qual será uma operação, porém automática. Segue na figura abaixo.

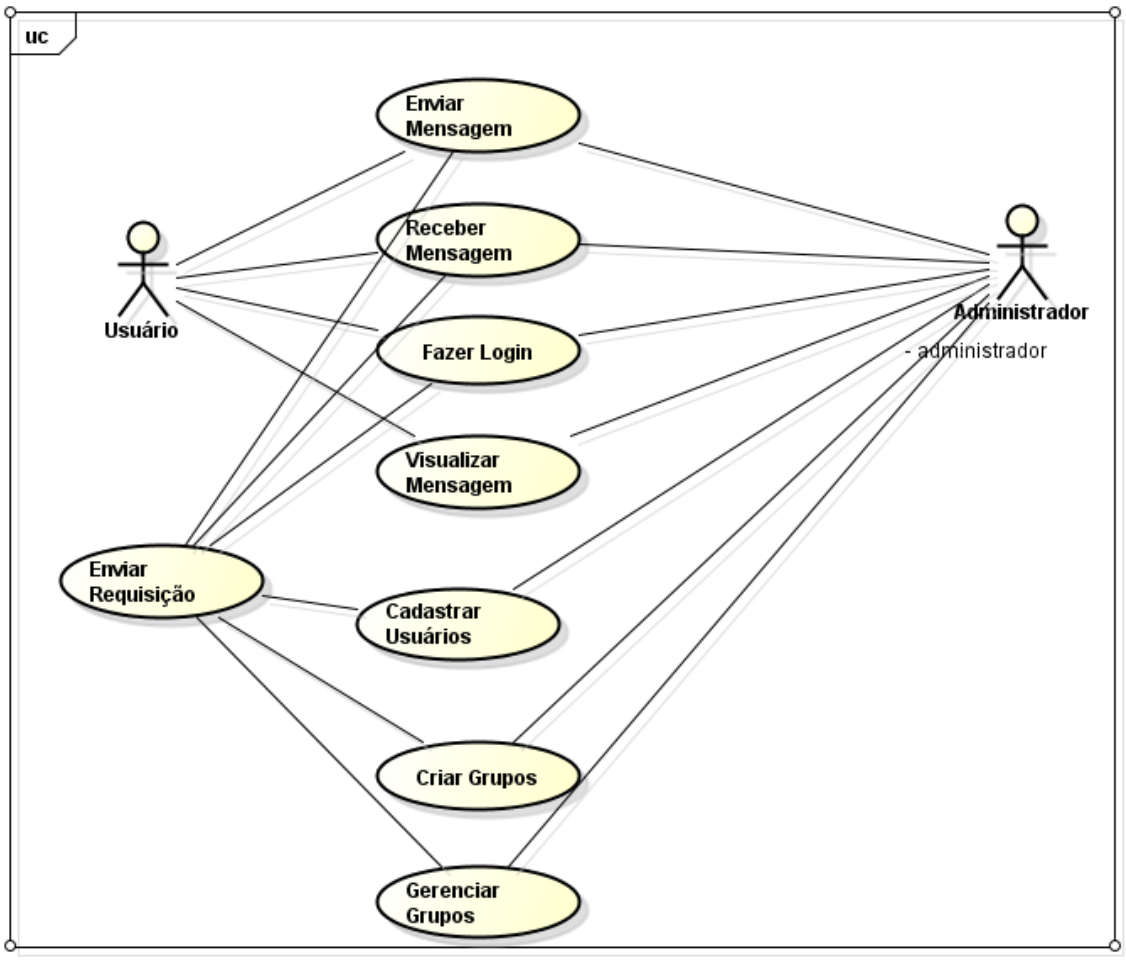

Figura 11 - Diagrama de Caso de Uso Geral

4.4 ADMINISTRADOR

A figura abaixo ilustra as iterações do administrador com o sistema e as ações que poderão ser realizadas.

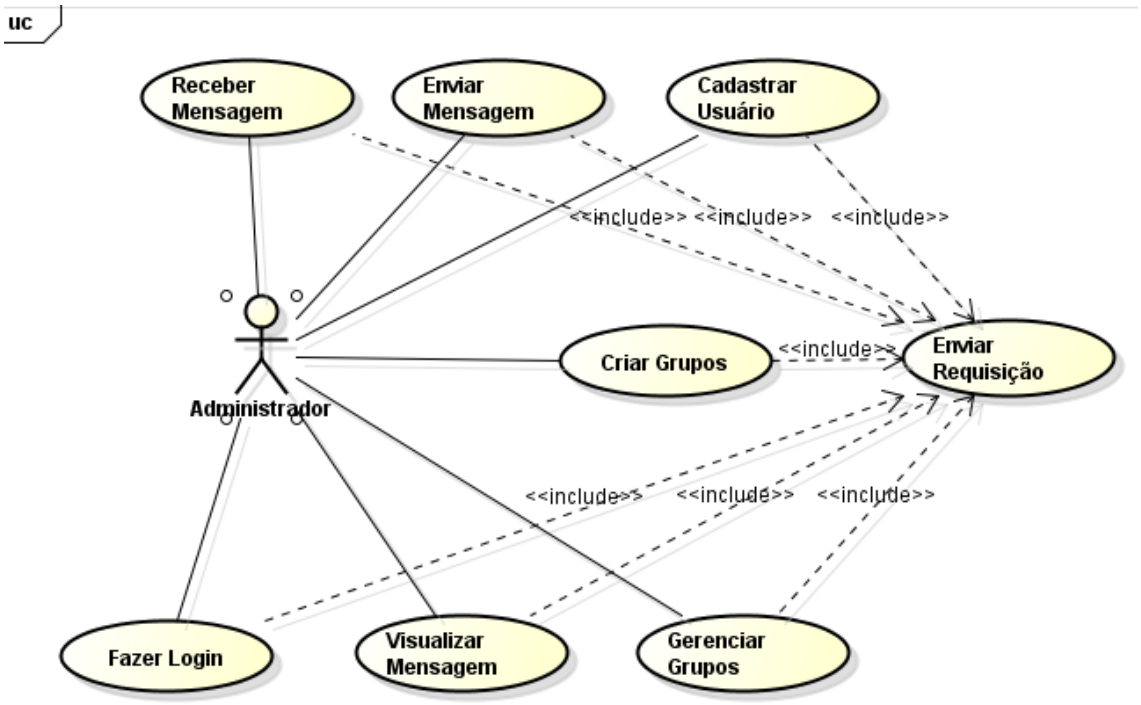

Figura 12 - Caso de Uso: Administrador

4.5 USUÁRIO

A figura abaixo ilustra as iterações do usuário com o sistema e as ações que deverão ser tomadas no uso da aplicação.

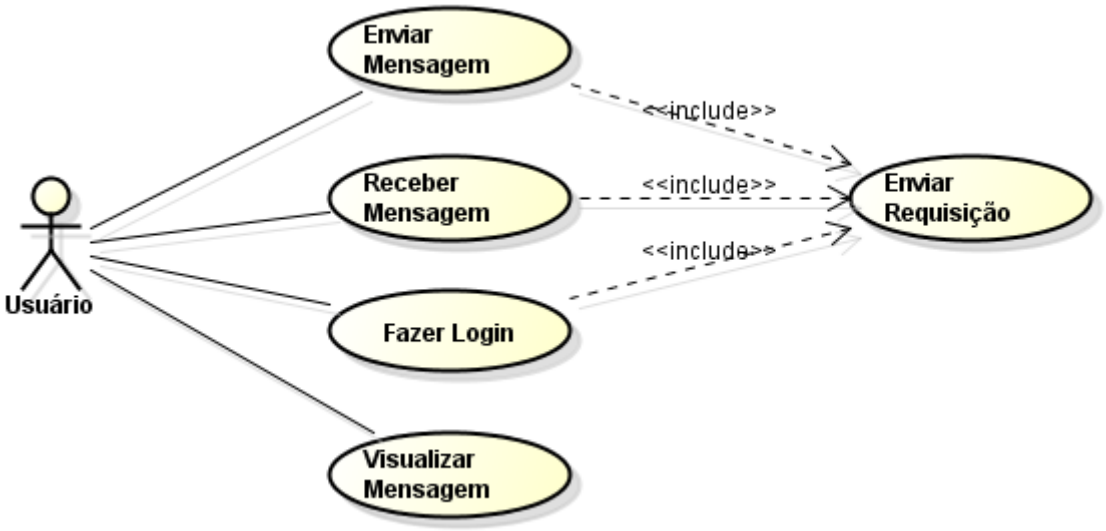

Figura 13 - Caso de Uso: Usuário

4.6 ESPECIFICAÇÃO DOS CASOS DE USO

A seguir serão apresentados os casos de uso específicos para cada iteração.

4.6.1 LOGIN

A figura abaixo exemplifica o caso de uso do Login (imagem da tela na página 58).

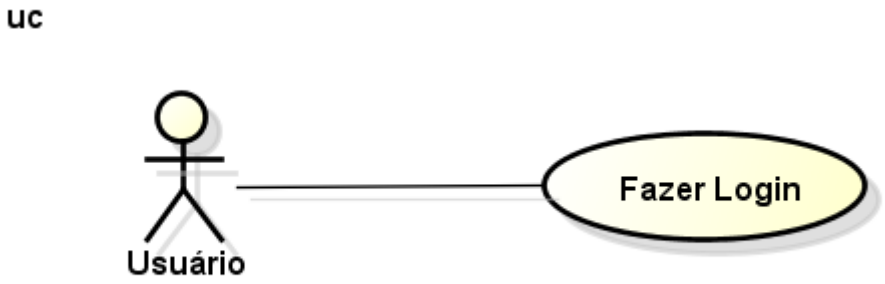

Figura 14 - Caso de Uso 1: Login

Finalidade/Objetivo:	Conectar o usuário ao sistema, para ter certeza de que ele pode ter acesso.
Ator (es):	Administrador e Usuário.
Pré-condições:	Já estar cadastrado no sistema.
Fluxo principal:	O usuário preenche os campos e pressiona o botão para fazer o login. O sistema fará uma requisição e validará o usuário e retornará uma mensagem de confirmação. Se houver dados incorretos ou falha durante a autenticação retornará uma mensagem de falha.

Tabela 5 - Narrativa do 1º Caso de Uso: Login

O usuário acessará o sistema e terá que fazer o login, quando ele acionar o botão para logar no sistema, os dados preenchidos serão enviados ao *REST criptografados*, onde será decriptado e analisado para detecção de possíveis

erros nos dados, caso tudo esteja correto o *REST* irá no banco resgatar os dados e conferir se ambos os dados estão iguais, se tudo estiver certo, o *REST* autorizará com que o usuário seja aceito pelo login. Caso qualquer uma destas operações cause algum tipo de erro, o *REST* irá responder imediatamente a aplicação desktop, que houve um erro durante o processo com uma mensagem especifica. Segue o diagrama na figura 15.

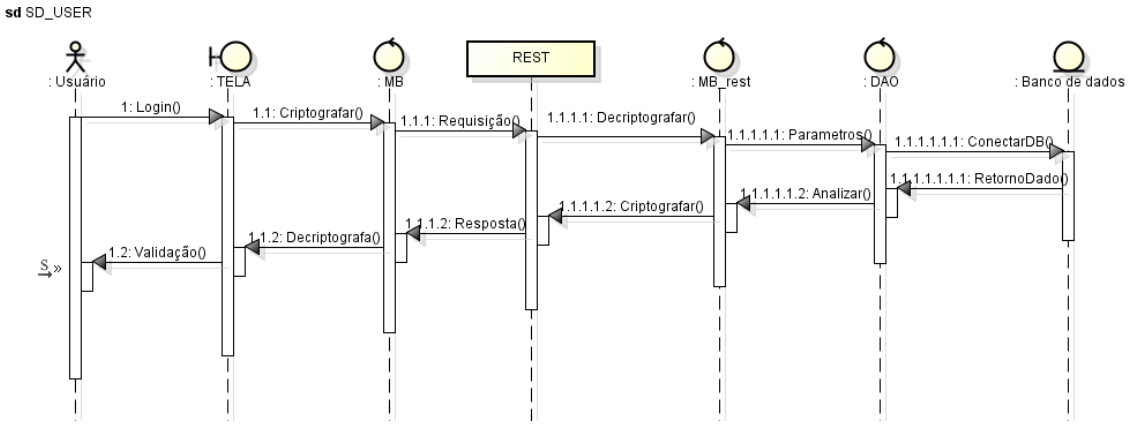

Figura 15 - Diagrama de Sequência 1: Login

4.6.2 ENVIO DE MENSAGEM

A figura abaixo exemplifica o caso de uso do envio de mensagem (imagem da tela na página 59).

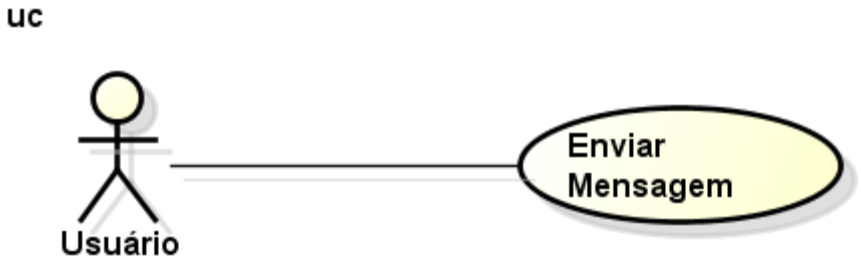

Figura 16 - Caso de Uso 2: Enviar de Mensagem

Finalidade/Objetivo:	Enviar mensagem para um grupo.
Ator (es):	Administrador e Usuário.
Pré-condições:	Estar logado no sistema e pertencer a um grupo.
Fluxo principal:	O usuário preenche os campos e pressiona o botão para enviar a mensagem. O sistema fará uma requisição para salvar os dados da mensagem no banco de dados. Se acontecer algum erro durante o processo uma mensagem de erro será retornada.

Tabela 6 – Narrativa do 2º Caso de Uso: Envio de Mensagem

Enviar mensagem fará com que a aplicação envie uma requisição para o REST, passando por parâmetro qual grupo aquela mensagem irá, quem está enviando e a mensagem. O REST analisará os dados e caso o usuário tenha acesso ao grupo o REST enviará os dados para serem salvos no banco de dados. Caso o banco insira corretamente, retornará uma mensagem ao REST dizendo que foi inserido com sucesso, com isso o REST retorna uma mensagem ao usuário dizendo que o processo foi bem sucedido. Segue na figura 17.

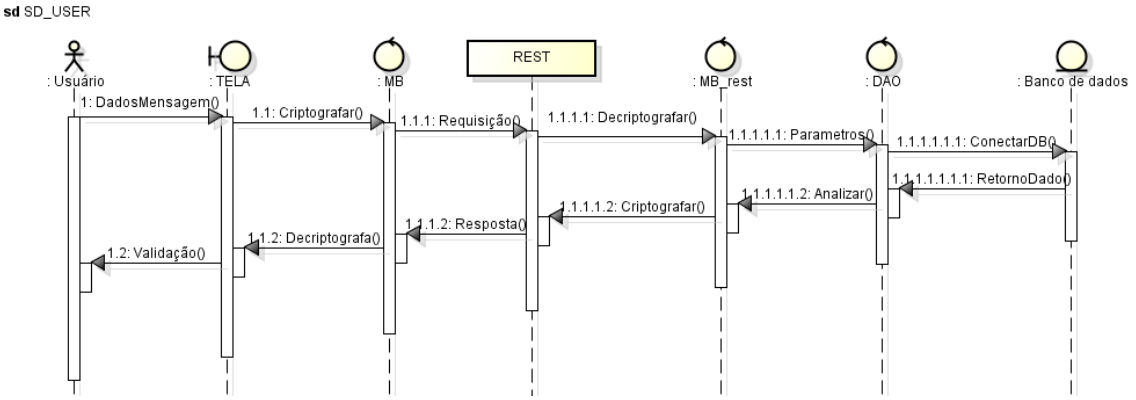

Figura 17 – Diagrama de Sequência 2: Envio de Mensagem

4.6.3 RECEBIMENTO DE MENSAGEM

A figura abaixo exemplifica o caso de uso do recebimento de mensagens .

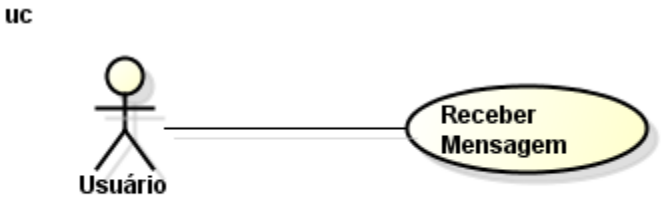

Figura 18 - Caso de Uso 3: Recebimento de Mensagem

Finalidade/Objetivo:	Receber mensagens do grupo.
Ator (es):	Administrador e Usuário.
Pré-condições:	Estar logado no sistema e pertencer a um grupo.
Fluxo principal:	A cada período de tempo o sistema fará uma requisição para obter novas mensagens.

Tabela 7 – Narrativa do 3º Caso de Uso: Recebimento de mensagem

Para receber mensagem o usuário não irá fazer nenhum estimulo, esta atividade será feita por um temporizador direto no sistema, onde a um determinado tempo fará uma requisição ao *REST* (3), ele confirmará se este usuário tem acesso ao grupo, caso tenha ele irá até o banco pegar as mensagens novas, o banco retornara a mensagem, e o *REST* entregará a mensagem ao usuário. Segue a figura 19.

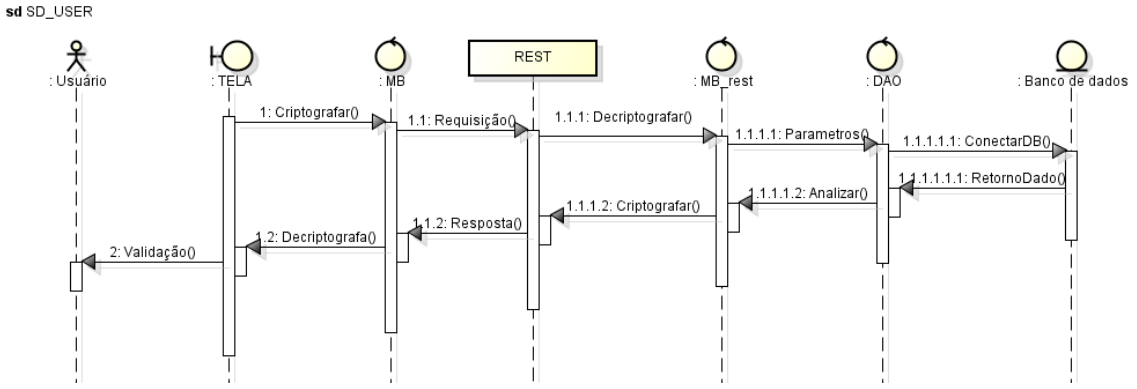

Figura 19 – Diagrama de Sequência 3: Recebimento de Mensagem

4.6.4 VISUALIZAÇÃO DE MENSAGEM

A figura abaixo exemplifica o caso de uso da visualização de mensagem.

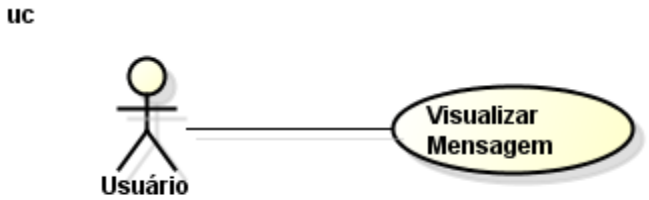

Figura 20 - Caso de Uso 4: Visualização de Mensagem

Finalidade/Objetivo:	Visualizar as mensagens do grupo.
Ator (es):	Administrador e Usuário.
Pré-condições:	Estar logado no sistema e pertencer a um grupo.
Fluxo principal:	Receber as mensagens. Abrir a aba de mensagens.

Tabela 8 – Narrativa do 4º Caso de Uso: Visualização de Mensagem

4.6.5 CADASTRO DE USUÁRIOS

A figura abaixo exemplifica o caso de uso de Cadastro de Usuários (imagem da tela na página 60).

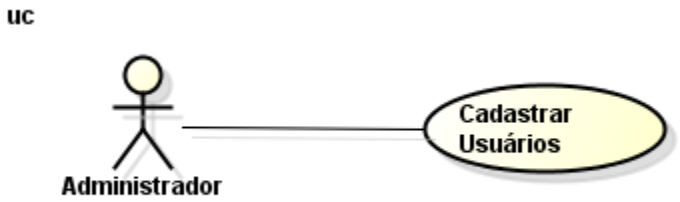

Figura 21 - Caso de Uso 5: Cadastro de Usuários

Finalidade/Objetivo:	Permitir inserção de novos usuários ou administradores.
Ator (es):	Administrador.
Pré-condições:	Estar logado no sistema e ser Administrador.
Fluxo principal:	Preenche os campos na tela e pressiona o botão cadastrar. Uma requisição será feita, e retornará uma mensagem de confirmação. Caso haja algum erro, uma mensagem de falha será o retorno.

Tabela 9 – Narrativa do 5º Caso de Uso: Cadastro de Usuários

Para cadastro de usuário, será enviada uma requisição ao *REST* e ele mandará os dados ao banco, para que possam ser salvos. O banco retornará uma confirmação à Web Service e depois ele retornará ao usuário a validação. Segue a figura 22.

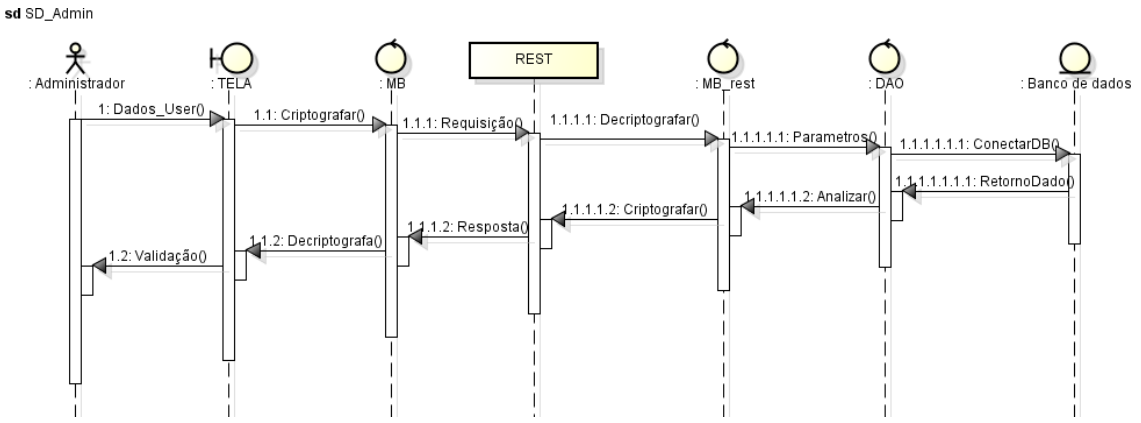

Figura 22 – Diagrama de Sequência 4: Cadastro de Usuário

4.6.6 CRIAÇÃO DE GRUPOS

A figura abaixo exemplifica o caso de uso de Criação de grupos (imagem da tela na página 61).

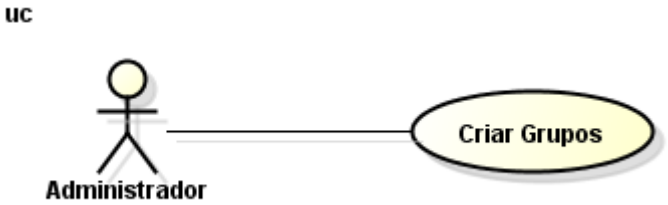

Figura 23 - Caso de Uso 6: Criar grupos

Finalidade/Objetivo:	Permitir a criação de grupos no sistema.
Ator (es):	Administrador.
Pré-condições:	Estar logado no sistema e ser Administrador
Fluxo principal:	Preenche os campos na tela e pressiona o botão cadastrar. Uma requisição será feita, e retornará uma mensagem de confirmação. Caso haja algum erro, uma mensagem de falha será o retorno.

Tabela 10 – Narrativa do 6º Caso de Uso: Criar grupos

Para criar um grupo o administrador irá enviar uma requisição ao REST, ele enviará os dados ao banco de dados, que retornará uma confirmação, e a Web Service retornará uma validação ao sistema desktop.

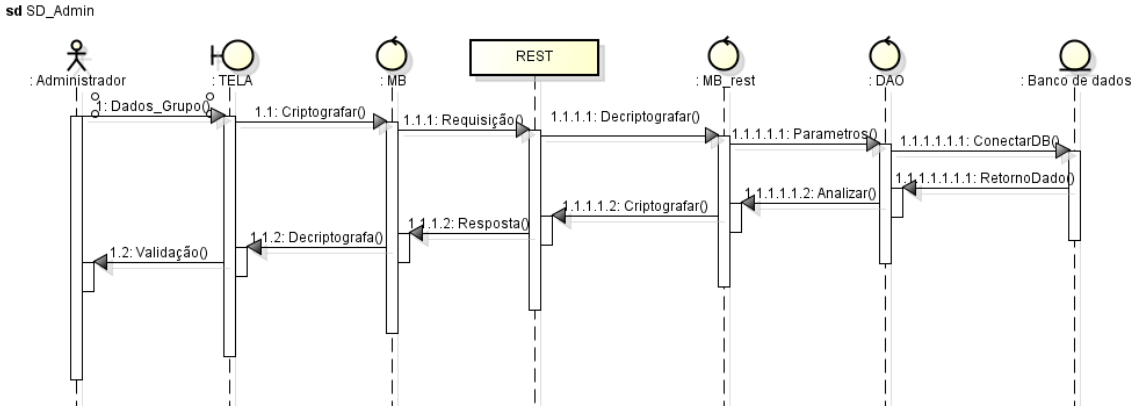

Figura 24 – Diagrama de sequência 6: Cadastro de Grupo

4.6.7 GERENCIAMENTO DE GRUPOS

A figura abaixo exemplifica o caso de uso de Gerenciamento de Grupos.

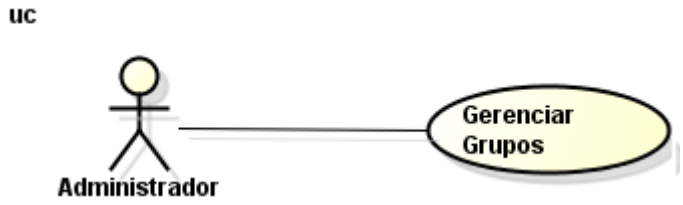

Figura 25 - Caso de Uso 7: Gerenciamento de Grupos

Finalidade/Objetivo:	Permitir o gerenciamento do grupo, inserir usuário no grupo, excluir do grupo.
Ator (es):	Administrador.
Pré-condições:	Estar logado no sistema e ser Administrador grupo especifica.
Fluxo principal:	Preenche os campos na tela e pressiona o botão enviar. Uma requisição será feita, e retornará uma mensagem de confirmação. Caso haja algum erro, uma mensagem de falha será o retorno.

Tabela 11 – Narrativa do 7º Caso de Uso: Gerenciamento de Grupos

Para gerenciamento de grupo o administrador enviará uma requisição ao *REST* para excluir ou adicionar alguém do grupo. A Web Service confirmará a se o administrar pode realizar a ação, caso sim, ele se conectará com o banco e salvará os dados. O banco retorna uma confirmação ao *REST*, que devolve ao sistema desktop. Segue a figura 25.

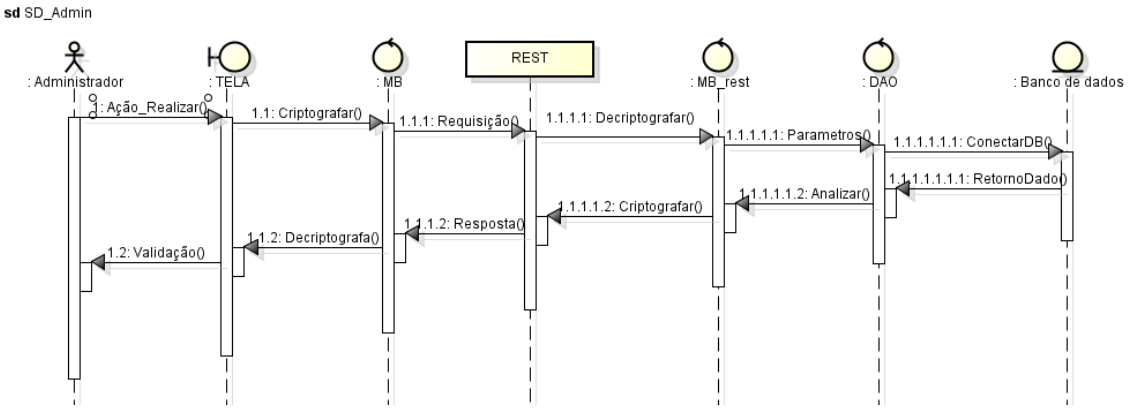

Figura 26 – Diagrama de Sequência 5: Gerenciamento de Grupo

4.7 DIAGRAMA DE CLASSE SISTEMA REST

A seguir são apresentados os diagramas de classes do sistema REST divididos em pacotes.

4.7.1 DIAGRAMA DE CLASSE DO PACOTE MODELO

A figura abaixo exemplifica o diagrama de classe do pacote modelo.

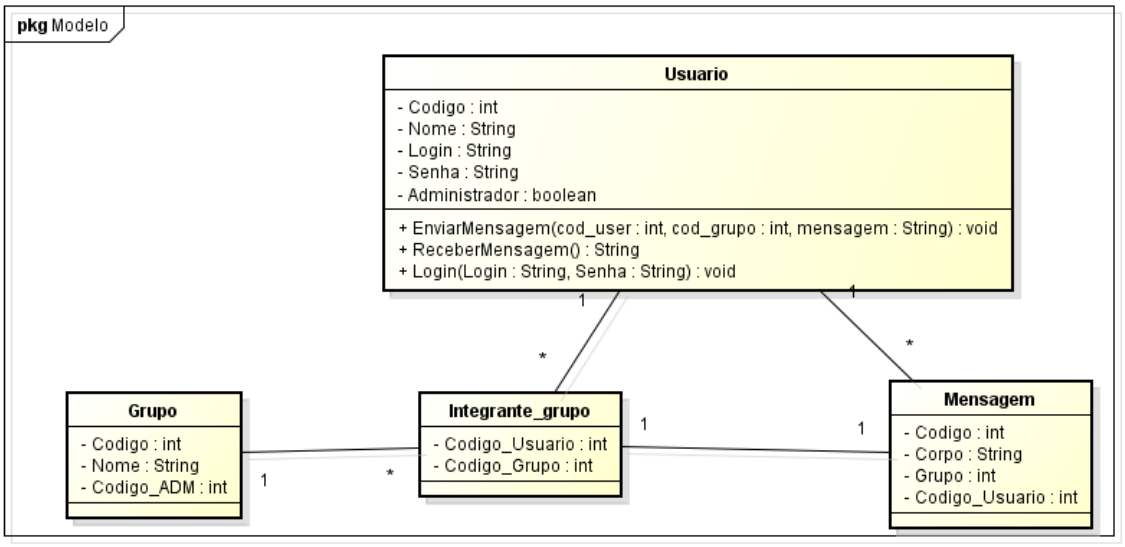

Figura 27 - Diagrama de classe pacote Modelo

4.7.2 DIAGRAMA DE CLASSE DO PACOTE UTIL

A figura abaixo exemplifica o diagrama de classe do pacote util.

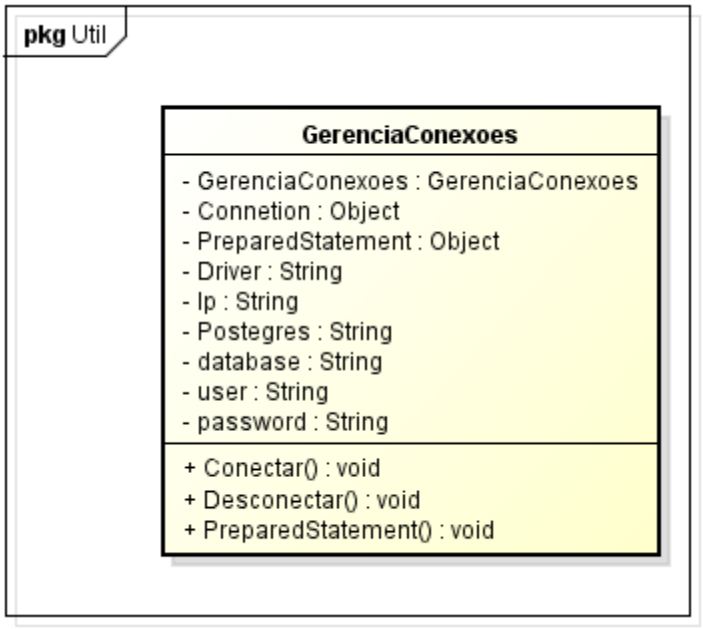

Figura 28 - Diagrama de classe pacote Útil

4.7.3 DIAGRAMA DE CLASSE DO PACOTE DAO

A figura abaixo exemplifica o diagrama de classe do pacote DAO.

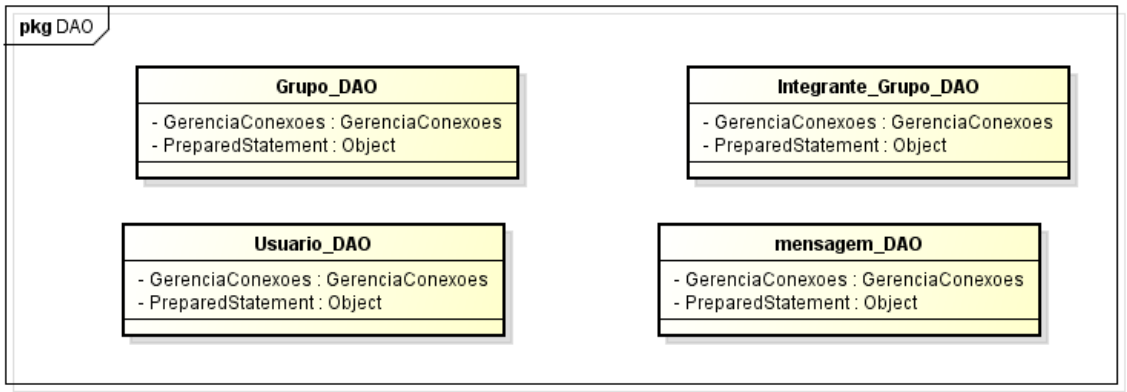

Figura 29 - Diagrama de classe pacote DAO

4.7.4 DIAGRAMA DE CLASSE DO PACOTE RESOURCE

A figura abaixo exemplifica o diagrama de classe do pacote resource.

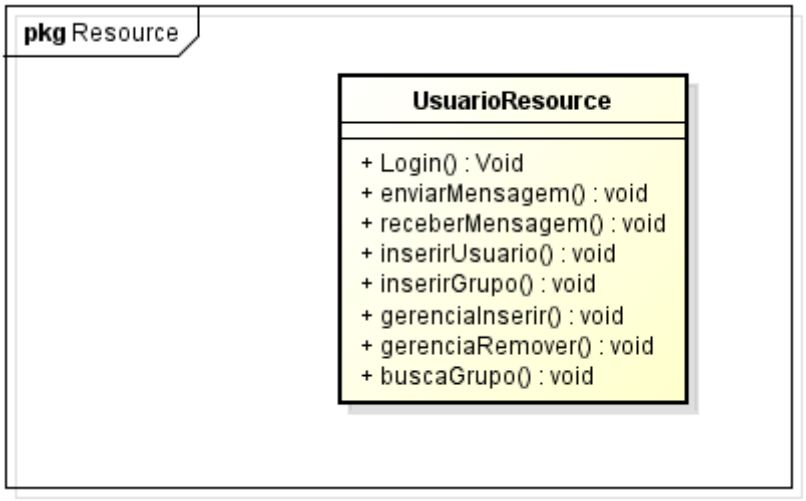

Figura 30 - Diagrama de classe pacote Resource

4.7.5 DIAGRAMA DE CLASSE DO PACOTE CONTROLLER

A figura abaixo exemplifica o diagrama de classe do pacote controller.

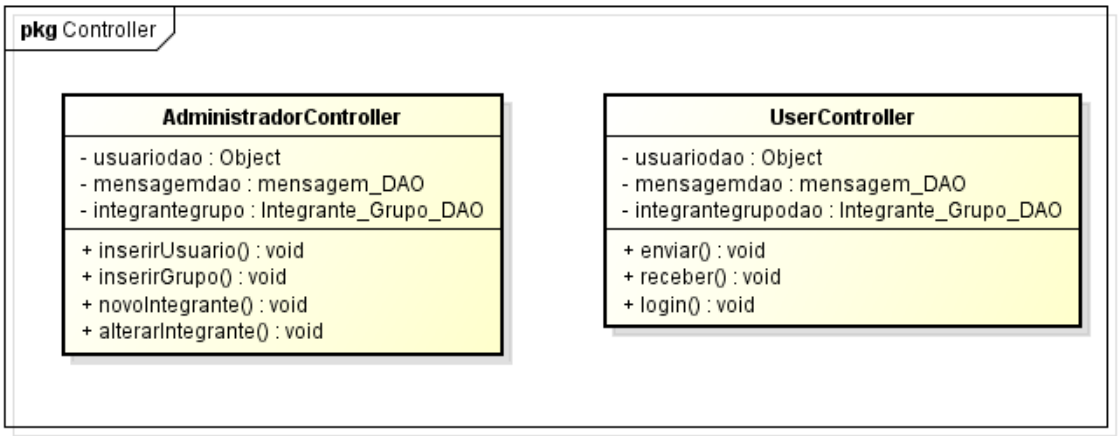

Figura 31 - Diagrama de classe pacote Controller

4.8 DIAGRAMAS DE CLASSE SISTEMA DESKTOP

A seguir temos os diagramas de classes do sistema desktop divididos em pacotes.

4.8.1 DIAGRAMA DE CLASSE DO PACOTE MODELO

A figura abaixo exemplifica o diagrama de classe do pacote modelo.

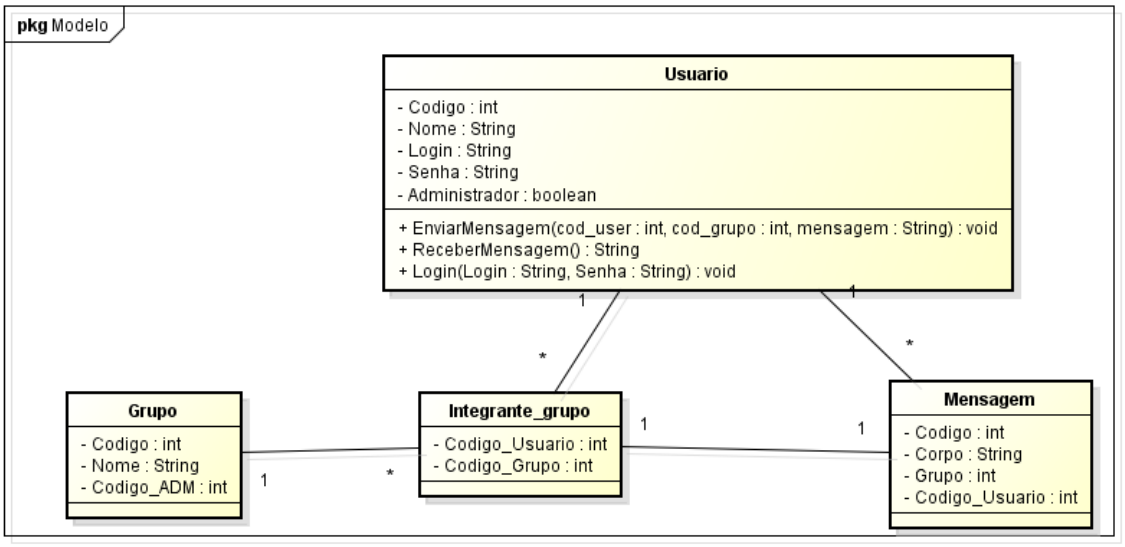

Figura 32 - Diagrama de classe pacote Modelo

4.8.2 DIAGRAMA DE CLASSE DO PACOTE TELAS

A figura abaixo exemplifica o diagrama de classe do pacote telas.

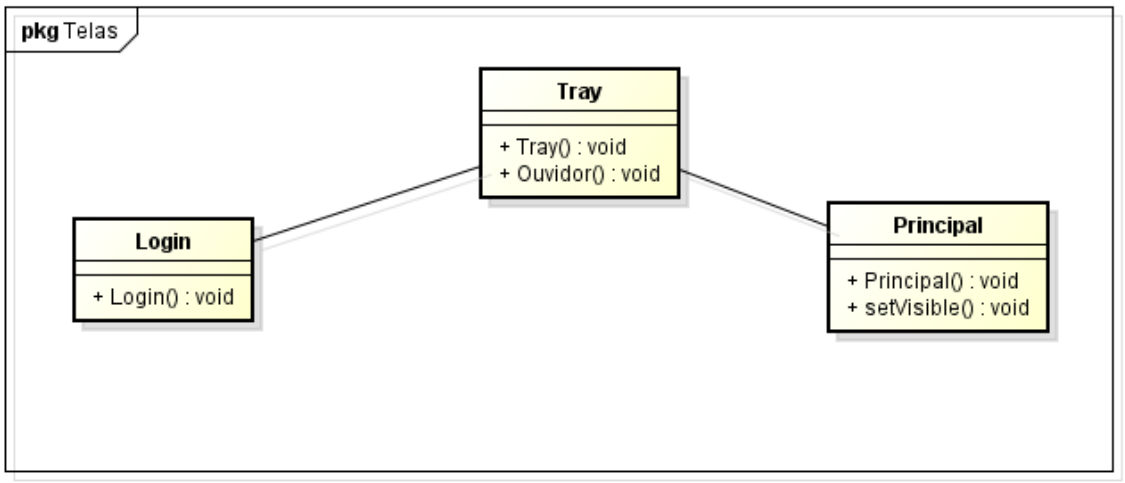

Figura 33 - Diagrama de classe pacote Telas

4.8.3 DIAGRAMA DE CLASSE DO PACOTE CONTROLLER

A figura abaixo exemplifica o diagrama de classe do pacote modelo.

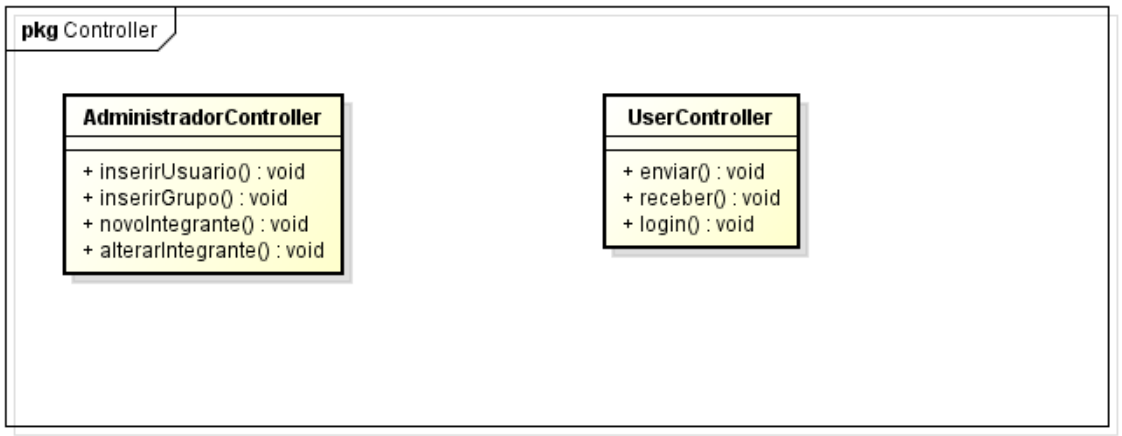

Figura 34 - Diagrama de classe pacote controller

4.9 DIAGRAMA DE ATIVIDADES

O diagrama de atividade está representado na figura abaixo.

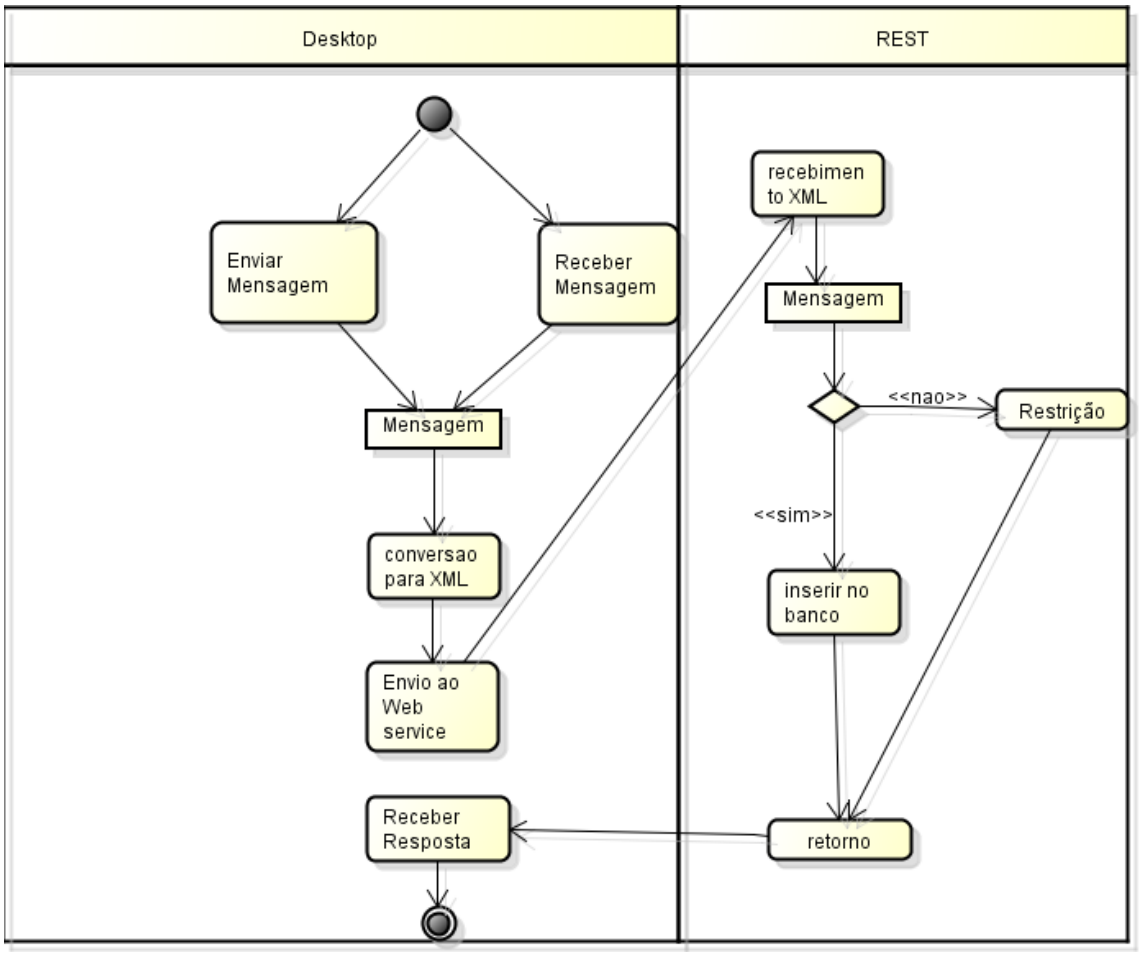

Figura 35 - Diagrama de Atividades

5 CONCLUSÃO

Após o encerramento de todas as etapas deste trabalho, foi possível realizar todos os objetivos propostos, como facilitar a integração entre aplicações distintas, efetuar autenticação do usuário por meio de um *WEB SERVICES*. Com a criação da aplicação *desktop* foi possível demonstrar os objetivos cumpridos e com a aplicação *REST* foi possível cumprir os objetivos.

O *WEB SERVICES* contém a autenticação do usuário, e a regra de negócio da aplicação *chat*. Ou seja caso haja a necessidade da criação deste mesmo *chat* em outra linguagem, só será preciso criar as telas, entrada de dados e o envio destes dados as *URL'S*, pois a regra de negócio está no *REST*.

O *REST* teve uma boa performance nas requisições e respostas a aplicação, com boa velocidade ao responder as requisições, com dados ou requisições erradas. Já o *chat* também teve uma boa performance levando as mensagens rapidamente aos usuários e sem problemas. O *chat* pode ser utilizado por qualquer empresa, visando, levar as mensagens para os diversos setores da mesma, além de ser possível de ser utilizado de qualquer lugar. Ele também facilitou a construção da aplicação *desktop,* pois para construí-la só foi preciso se preocupar com a parte de entrada de dados, e o envio destes dados as *URL's* corretas. A parte mais complexa dos sistemas que normalmente é a regras de negócio, ficou dentro do *REST*, assim uma camada teórica de segurança foi adicionada, além das outras como a cifra dos dados e autenticação do usuário.

Este trabalho, foi de grande importância para esclarecer sobre a viabilidade da utilização do *REST.* Onde foi possível demostrar que ele obteve uma boa performance ao número de requisições respondidas e recebidas, obteve mais segurança entre as aplicações, um bom gerenciamento de usuários ativos no sistema e sua facilidade em integração entre sistemas.

REFERÊNCIAS BIBLIOGRÁFICAS

Dicionário Aurélio, Coordenação Marina Baird Ferreira. Editora: Editora Positivo, 8ª edição, impressão 2013.

DUMAS, Véronique. A origem da internet. Disponível em: http://www2.uol.com.br/historiaviva/reportagens/o_nascimento_da_internet.html . Acessado em 01/09/2014.

GURUGE, Anura. Web Services: Theory and Practice, First Edition. ELSEVIER Digital Press, 2004

KALIM, Martin. Java Web Services Implementando, Tradução: Raquel Marques Primeira Edição. Editora: Alta Books, 2010.

Percentual de pessoas que acessaram a internet na população de 10 anos ou mais de idade. Disponível em: http://www.ibge.gov.br/home/presidencia/noticias/imprensa/ppts/00000012962305122013234016242127.pdf. Acessado em 30/08/2014.

SAUDATE, Alexandre. Rest construa API's inteligentes de maneira simple. Editora Casa do Código, 2013b.

SAUDATE, Alexandre. SOA aplicado integrando com web service e além. Editora Casa do Código, 2013a.

Tabela de Salários Digitais. Disponível em: http://info.abril.com.br/carreira/salarios/ .Acessado em 10/03/2015.

TERADA, Routo. segurança de dados: Criptografia em rede de computador, 2 ª Edição Blucher , 2008.

TITTEL, ED. Teoria e Problemas de XML. Tradução de Ralph Miller Jr. Editora: Bookman, 2002.

W3C. SOAP Version 1.2. Disponível em: http://www.w3.org/TR/soap12-part1. Acessado em 01/09/2014a.

W3C. Web Semântica . Disponível em: http://www.w3c.br/Padroes/WebSemantica. Acessado em 1/11/2014b.
Internet Users in the world Distribution by World Regions-2013 Q4. Disponível em: http://www.internetworldstats.com/stats.htm Acessado em 10/11/2014.

.

ANEXO I – TELAS DO SISTEMA

A seguir são apresentadas as telas do sistema desenvolvido.

TELA DE LOGIN

A figura abaixo apresenta a tela inicial para acesso ao sistema, onde o usuário informa seus dados de acesso (Login e Senha).

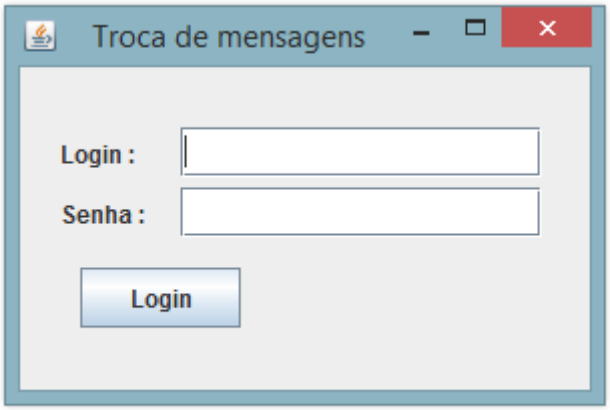

Figura 36 - Tela de Login

TELA DE MENSAGENS

A Figura a seguir apresenta a tela de mensagens, responsável por enviar e receber mensagens, ela também contém botões que fazem parte da administração onde somente usuários administradores terão acesso e visão, indicado na figura com um retângulo vermelho.

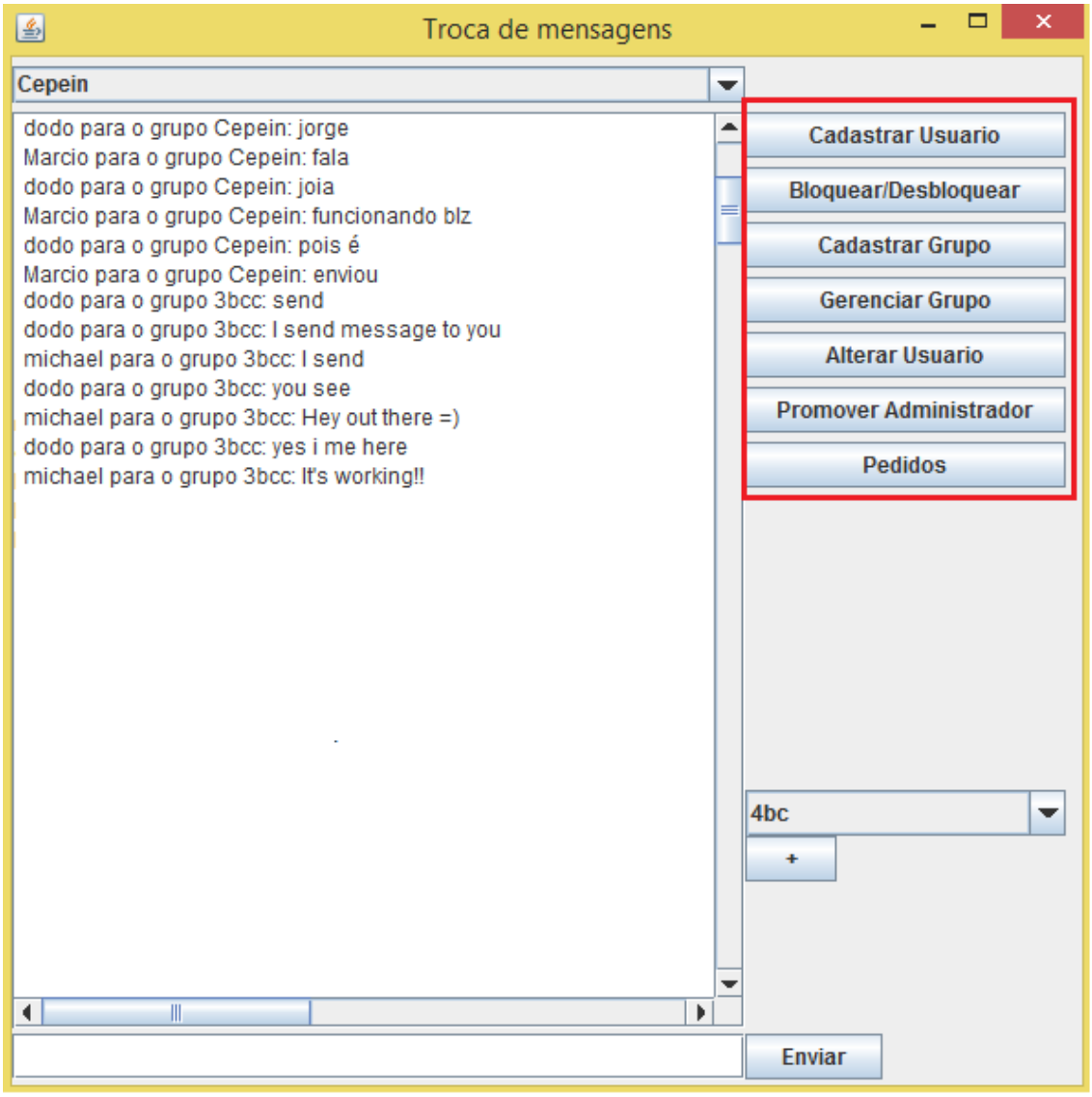

Figura 37 – Tela de mensagem

TELA DO SYSTEM TRAY

A figura abaixo ilustra um system tray, responsável por deixar a aplicação ativa e de forma visível.

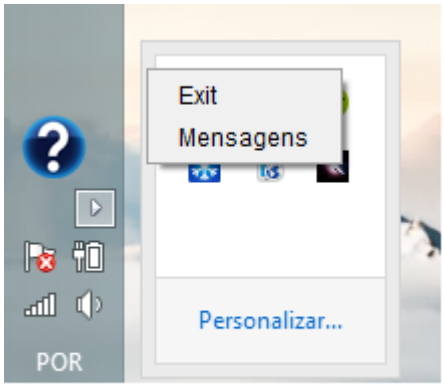

Figura 38 – Tela do System Tray

TELA DE CADASTRO DE USUÁRIO

A figura abaixo representa a tela de cadastro de usuário, permitindo inserir novos usuários ao sistema, de acordo com os campos.

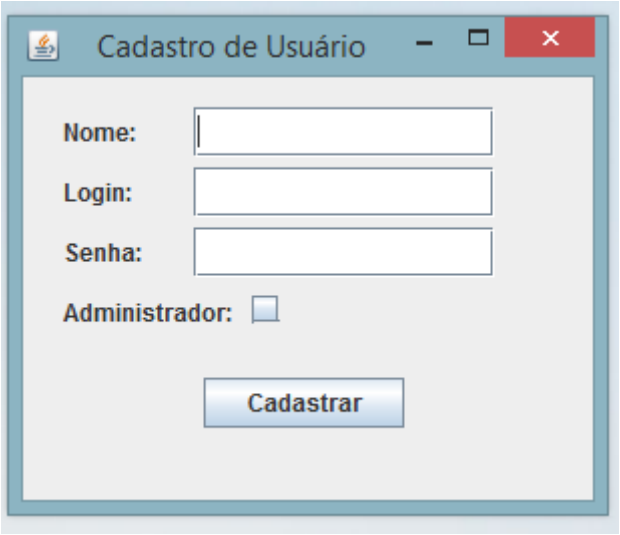

Figura 39 – Tela de Cadastro de Usuário

TELA DE CADASTRO DE GRUPO

A figura abaixo representa a tela de criação de um novo grupo.

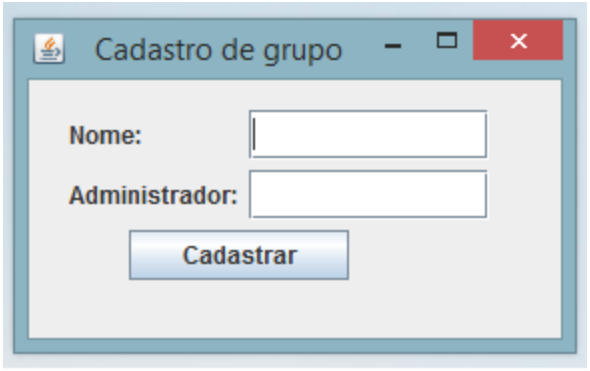

Figura 40 – Tela de Cadastro de Grupo

TELA DE BLOQUEIO DE USUÁRIO

A figura abaixo é responsável pela tela de bloquear usuário no grupo. Onde o botão busca traz todos os usuários do grupo determinado pela primeira ComboBox, e a segunda ComboBox é carregado com todos os usuários deste grupo.

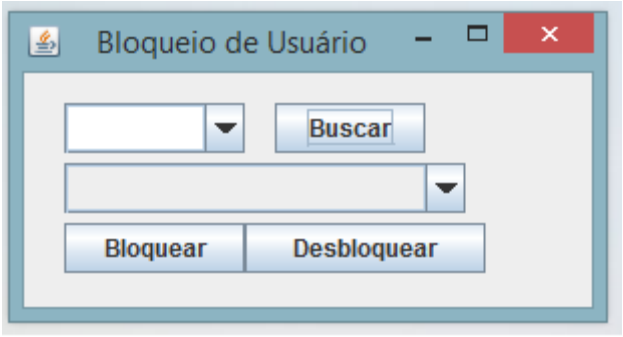

Figura 41 – Tela de Bloqueio de Usuário

TELA DE USUÁRIO GRUPO

A figura abaixo é responsável por inserir um usuário em um grupo. Onde é carregado os grupos na primeira ComboBox, a segunda é carregada com todos os usuários que não fazem parte do grupo, a terceira ComboBox serve para armazenar os usuários que deseja inserir no grupo, podendo ser somente um ou vários. Os botões (<,>) são para inserir ou remover usuários da lista de inserção.

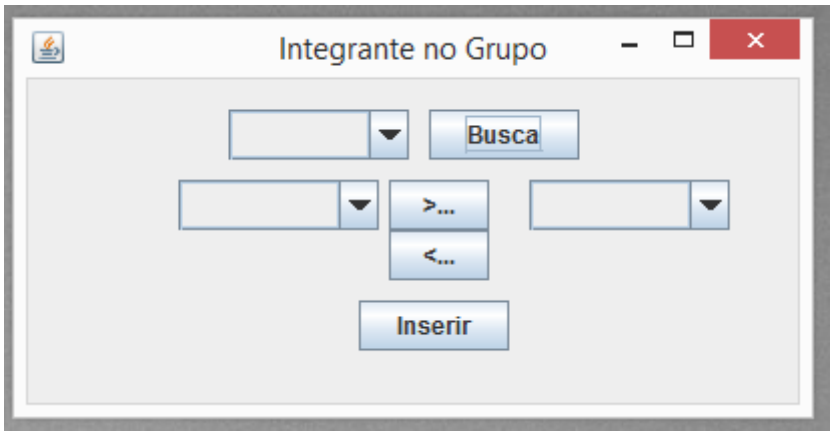

Figura 42 – Tela de Usuário Grupo

TELA DE ALTERAÇÃO DE USUÁRIO

A figura abaixo representa a tela para permitir que um usuário possa ser administrador.

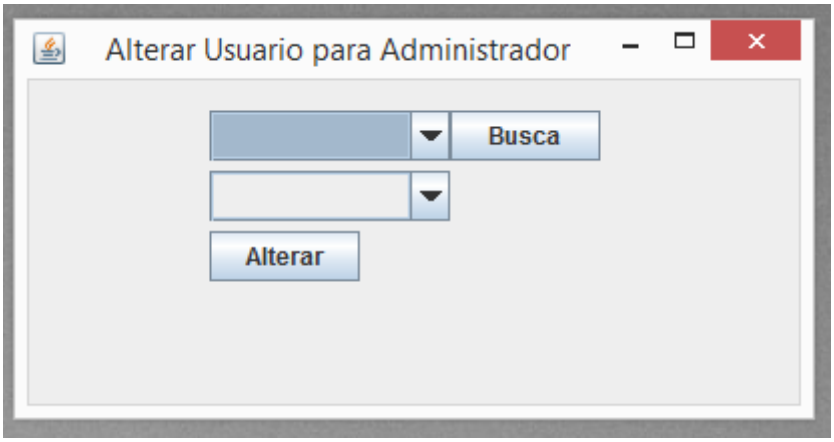

Figura 43 – Tela de Alteração de Usuário

TELA DE PROMOÇÃO A ADMINISTRADOR

A figura abaixo representa a tela para promover um usuário a administrador.

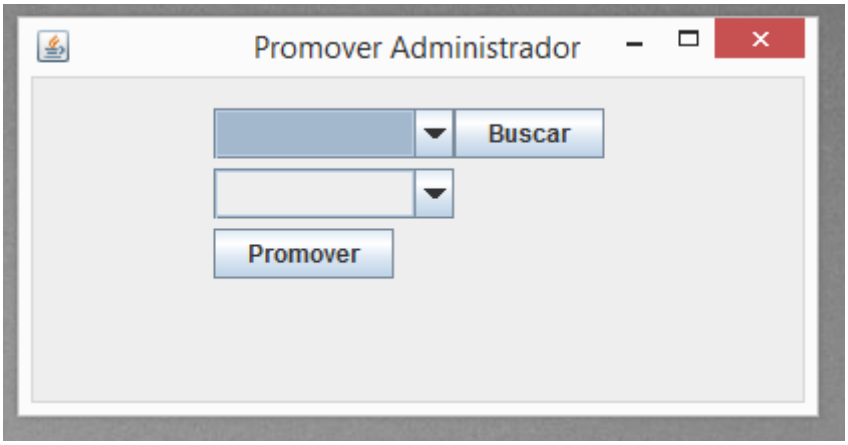

Figura 44 – Tela de Promoção a Administrador

TELA DE PEDIDOS

A figura abaixo representa a tela de pedidos de grupo, onde aparecerão os usuários que pediram para participar do determinado grupo.

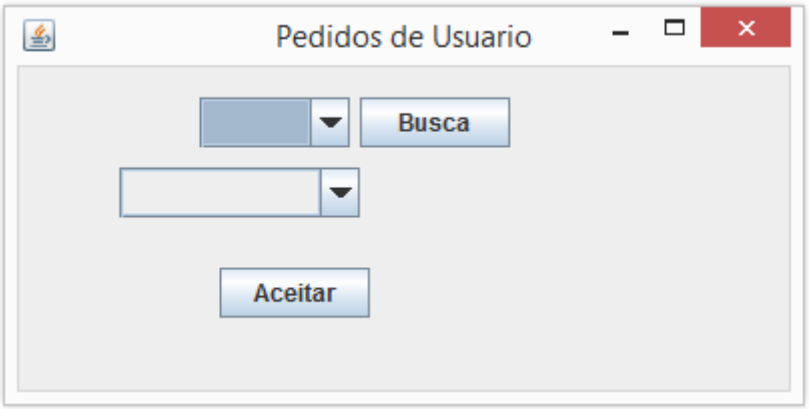

Figura 45 – Tela de Pedidos

ANEXO II – CÓDIGOS FONTES DOS SISTEMAS

MÉTODO PARA CRIAÇÃO DE ARQUIVO

A figura abaixo apresenta o método para criar um arquivo ou caso ele já exista adicionará novos dados no fim do mesmo.

```
/**
 * cria um arquivo se nao existir e insere os dados , caso exista somente
 * adicionara
 *
 * @param nome nome do arquivo que vai criar
 * @param dados dados que serao inseridos no arquivo
 */
public void criarArquivo(String nome, String dados) {
    File file = new File(nome);
    if (!file.exists()) {
        try {
            file.createNewFile();
            FileWriter xml = new FileWriter(nome);
            xml.write(dados);
            xml.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } else {
        FileWriter xml;
        try {
            xml = new FileWriter(nome, true);
            xml.write(dados);
            xml.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }
}
```

Figura 46 - Código fonte do método: Criar Arquivo

MÉTODO RESPONSAVEL PELA REQUISIÇÃO GET

A figura abaixo apresenta o método para enviar uma requisição do tipo GET.

```
/**
 * GET alterar usuario adm
 *
 * @param id grupo
 * @return inteiro se deu certo ou nao
 */
public int alterUser(int id) {
    String[] result = new String[2];

    try {
        String urla = ini;
        urla += id;
        urla += "/admalt";
        URL url = new URL(urla);
        HttpGet httpget = new HttpGet(urla);
        HttpResponse response;
        response = HttpClientSingleton.getHttpClientInstace().execute(httpget);
        HttpEntity entity = response.getEntity();

        if (entity != null) {
            result[0] = String.valueOf(response.getStatusLine().getStatusCode());
            InputStream instream = entity.getContent();
            result[1] = toString(instream);
        }

    } catch (IOException io) {
        io.printStackTrace();
    }
    return Integer.parseInt(result[0]);
}
```

Figura 47 – Código fonte do método: Requisição GET

MÉTODO RESPONSAVEL PELA REQUISIÇÃO POST

A figura abaixo apresenta o método para enviar uma requisição do tipo POST.

```
/**
 * POST
 *
 * @param ig objeto de integrante grupo
 * @param cod do administrador
 * @return inteiro se deu certo ou nao
 */
public int bloquearUsuario(Integrante_Grupo ig, int cod) {
    int retorno = 0;
    try {
        String urla = ini;
        urla += "adm/";
        urla += cod;
        urla += "/gerencia/remover";
        URL url = new URL(urla);
        String t1 = Conversao.ObjectforString(ig);
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("POST");
        con.setRequestProperty("Content-Type", "application/xml");
        con.setDoInput(true);
        con.setDoOutput(true);
        OutputStreamWriter out = new OutputStreamWriter(con.getOutputStream());
        out.write(t1);
        out.flush();

        retorno = con.getResponseCode();
        con.disconnect();

    } catch (IOException io) {
        io.printStackTrace();
    }
    return retorno;
}
```

Figura 48 – Código fonte do método: Requisição POST

MÉTODO DE RESOURCES DA WEB SERVICE

A figura abaixo apresenta o método dos resources da web service, responsável por fazer validação, chamada da camada de CONTROLLER e resposta ao cliente.

```
@POST
@Path("/{id}/enviar")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public Response enviarMensagem(Mensagem mensagem, @PathParam("id") Integer id) {

    if (mensagem != null) {
        UsuarioController usuarioC = new UsuarioController();
        if (usuarioC.enviar(mensagem)) {

            return Response.status(200).build();// sucesso
        } else {
            Response.status(500).build(); // erro na autenticacao ou dados incorretos
        }
    }
    return Response.status(400).build(); // objeto nulo

}

@GET
@Path("/{id}/{cod}/receber")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public List<Mensagem> receberMensagem(@PathParam("id") Integer id, @PathParam("cod") Integer cod) {
    List<Mensagem> lst = new ArrayList<>();

    UsuarioController usuarioC = new UsuarioController();
    lst = usuarioC.receber(cod, id);
    return lst;

}
```

Figura 49 – Código fonte dos métodos GET E POST (servidor)

MÉTODO THREAD

A figura abaixo apresenta um código com um thread, responsável por fazer uma busca no banco a cada intervalo de tempo determinado para trazer novas mensagens.

```
public void ouvidor(final List<Integrante_Grupo> lst, final int id) {
    long TEMPO = (1000 * 1);
    final String opcao[] = {"Agora", "Mais tarde"};
    Timer timer = null;
    if (timer == null) {
        timer = new Timer();
        TimerTask tarefa = new TimerTask() {
            public void run() {
                try {
                    teste = false;
                    UsuarioController uc = new UsuarioController();
                    for (int i = 0; i < lst.size(); i++) {
                        teste = uc.carregarMensagems(lst.get(i).getGrupo().getCodigo(), id);
                    }
                    mensagens.setText(uc.abrirMensagens());
                    if (teste) {
                        mensagens.setText(uc.abrirMensagens());
                        int op = JOptionPane.showOptionDialog(rootPane, "Você tem mensagen. Deseja abrir ?",
                                "Nova mensagem", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null,
                                opcao, rootPane);
                        if (op == 0) {
                            teste = false;
                            visibilidade();
                        }
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
        timer.scheduleAtFixedRate(tarefa, TEMPO, TEMPO);
    }
}
```

Figura 50 – Código fonte do método de thread