PROTÓTIPO DE SISTEMA DE TROCA DE MENSAGENS EM DELPHI BASEADO EM APACHE ACTIVEMQ

Bruna Luisa Gessner, Mauro Marcelo Mattos – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

bru_gessner@yahoo.com.br, mattos@furb.br

Resumo: *Sistemas de mensagens são tecnologias que permeiam a sociedade atualmente pois permitem a troca de mensagens em alta velocidade e de forma assíncrona entre sistemas diferentes. Há servidores e clientes desenvolvidos em várias linguagens de programação, e, em Delphi há duas soluções disponíveis no mercado: uma proprietária e uma disponibilizada em código aberto. Este trabalho descreve um protótipo de aplicação que permite a troca de mensagens entre os usuários conectados utilizando esta biblioteca de código aberto. O desenvolvimento é baseado na biblioteca Stomp disponibilizada como software open source e no servidor Apache ActiveMQ. São apresentados a fundamentação teórica, a especificação e as principais características do protótipo. Como conclusões é possível afirmar que o projeto permite a troca de mensagens e o envio de imagens em chat privado e de grupo. Para os usuários em chat privado é permitido ainda a consulta do histórico de conversas trocadas com cada usuário.*

Palavras-chave*: Apache ActiveMQ. Mensageria. Delphi XE.STOMP.MOM.*

1 INTRODUÇÃO

Atualmente, um dos aspectos mais notáveis observados na sociedade da informação é a convergência tecnológica dos meios de comunicação, através de um longo processo de adaptação de seus recursos comunicativos às mudanças evolutivas (TEIXEIRA, 2012). A palavra comunicação tem origem no Latim *Communicatio* que significa ação de tornar algo comum a muitos (POYARES, 1970).

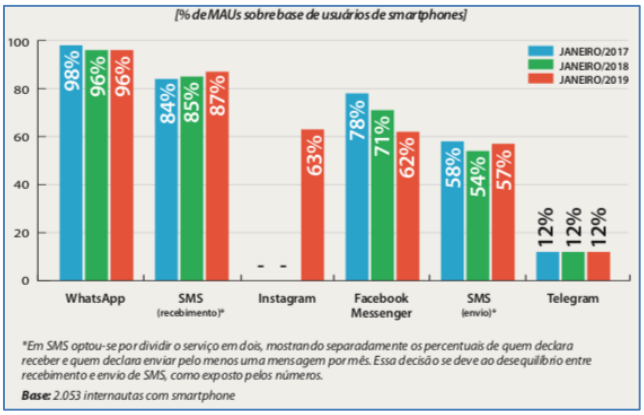
> No final do século XX, já era prática comum as pessoas estarem ligadas à Internet e comunicarem entre si utilizando diversas aplicações clientes de *instantmessaging* (IM). IM fornece capacidades de comunicação entre utilizadores quase em tempo real além de permitir visualizar a informação relacionada com a presença de um utilizador (se está online, offline, etc.). No passado existiam muitas aplicações de clientes conectadas a diferentes servidores que implementavam também protocolos de comunicação proprietários distintos. Na prática, a utilização de protocolos proprietários distintos leva a que os utilizadores de uma aplicação de IM não possam comunicar com os utilizadores de outra aplicação caso esta utilize um protocolo distinto. (REIS, 2012,p.51)

Os meios de comunicação estão se fazendo cada vez mais presentes na sociedade e os dispositivos móveis estão ocupando um espaço enorme. Novos aplicativos surgem todos os dias com o intuito de facilitar ainda mais o dia a dia do usuário, onde é possível afirmar que a sociedade está vivendo a "era dos aplicativos" (BRAZAN, 2012). Conforme Paiva (2019) a pesquisa "Panorama MobileTime/Opinion Box" sobre mensageria no Brasil realizou as seguintes descobertas:

a) está crescendo o uso de WhatsApp para comunicação entre marcas e consumidores;
b) o Instagram está presente em 65% dos smartphones brasileiros e o perfil médio de sua base é jovem e feminina;
c) a base de usuários do Messenger está se erodindo e aqueles que ficam estão usando cada vez menos o App;
d) a utilização do SMS está aumentando, tanto para envio quanto para recebimento de mensagens de texto.

A Figura 1 apresenta um gráfico demonstrando o uso da tecnologia considerando a proporção de usuários ativos mensais ou MAU (na sigla em inglês). A pergunta realizada foi: Marque as formas de comunicação que você utiliza no WhatsApp/Facebook/Messenger/Telegram – Pode marcar mais de uma. Conforme o autor, responderam a esta pergunta: 1975 usuários de WhatsApp, 1268 usuários de Facebook Messenger e 246 usuários de Telegram.

Figura 1– Evolução da popularidade de serviços de mensageria móvel no Brasil



Fonte: Paiva (2019).

A Figura 2 apresenta um aspecto muito relevante na pesquisa que se refere às formas de comunicação e a proporção por tipo de conteúdo com que são utilizadas nos aplicativos WhatsApp/Facebook Messenger/Telegram. A pergunta realizada foi: "Marque as formas de comunicação que você utiliza no WhatsApp/Facebook Messenger/Telegram - Pode marcar mais de uma". Conforme o autor, responderam a esta pergunta: 1985 MAUs de Whatsapp, 1268 MAUs de Facebook Messenger e 246 MAUs de Telegram.

Figura 2- Proporção de uso por tipo de conteúdo trafegado em cada mensageiro (% sobre MAUs)

	WhatsApp	Facebook Messenger	Telegram
Troca de mensagens de texto	92%	83%	74%
Troca de imagens	77%	47%	49%
Troca de mensagens de áudio	77%	28%	42%
Troca de vídeos	65%	27%	42%
Chamadas de voz	62%	18%	30%
Desenhos (emojis)	59%	34%	32%
Videochamadas	50%	17%	N.D.
Mensagens efêmeras	43%	17%	N.D.
Troca de mensagens de vídeo gravadas dentro do app	N.D.	N.D.	28%

Fonte: Paiva (2019).

Um outro aspecto interessante da pesquisa refere-se à frequência de uso de cada aplicativo (Figura 3). A pergunta realizada foi: "Pensando nos últimos meses, com que frequência você abre o WhatsApp/Facebook Messenger/Instagram/Telegram para ler ou enviar mensagens?". Conforme o autor, responderam a esta pesquisa: 1998 internautas de WhatsApp, 1413 com Facebook Messenger, 1331 usuários de Instagram e 271 usuários de Telegram.

Figura 3- Frequência de uso de cada aplicativo

	Todo dia	Quase todo dia	Algumas vezes por semana	Algumas vezes por mês	Quase nunca	Nunca
WhatsApp	91%	6%	2%	0,4%	0,4%	0,2%
Facebook Messenger	39%	23%	20%	8%	9%	1%
Instagram	66%	18%	10%	3%	2%	1%
Telegram	35%	28%	20%	7%	8%	2%

Fonte: Paiva (2019).

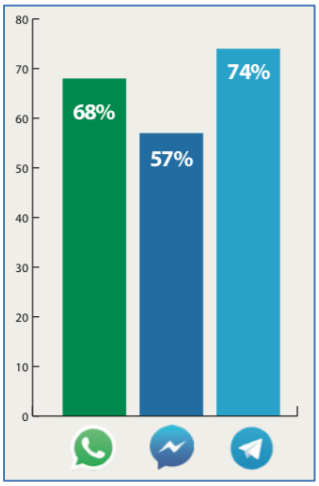
Neste contexto de proliferação de uso de serviços de mensagens, cada vez mais as empresas buscam disponibilizar soluções que façam uso de plataformas móveis para realizar negócios. Pacete (2019,p.49) afirma que:

> Os últimos meses não foram tão positivos para a reputação de aplicativos de mensagens como o WhatsApp. A onda de fake News e o protagonismo do serviço nas eleições ofuscaram, de certa forma, o potencial da mensageria para os negócios. O levantamento Mobile Time/Opinion Box, feito com 1984 brasileiros com acesso à internet, ao longo de agosto de 2018, mostrou que 53% dos internautas que possuem uma conta no WhatsApp utilizariam a ferramenta para pagamentos e transferências. De acordo com a pesquisa, esse número equivale a 50 milhões de pessoas se os dados da pesquisa forem cruzados com os do IBGE e do estudo TIC Domicílios.

A partir da liberação oficial do WhatsApp como canal de comunicação entre grandes marcas e seus consumidores, realizada em agosto de 2018, a proporção de usuários ativos mensais que conversam com marcas dentro do app passou

de 55% para 63% em seis meses. No Facebook Messenger, de 51% para 57%. E no Telegram, de 48% para 55%. (PACETE; 2019). Paiva (2019) apresenta na Figura 4 o resultado da pergunta: Você se comunica com marcas e empresas através do WhatsApp/Facebook Messenger/Telegram?. Conforme o autor, responderam esta pergunta: 1975 usuários de WhatsApp, 1268 usuários de Facebook Messenger e 246 usuários de Telegram.

Figura 4 – Proporção de usuários que se comunicam com marcas pelo App



Fonte: Paiva (2019).

A partir das estatísticas apresentadas, fica evidenciado que o tema mensageria é um tema relevante e atual a ser estudado. Tendo em vista a experiência da autora com Delphi e considerando que atualmente o ambiente apresenta-se ao mercado como sendo possível o desenvolvimento de aplicações multiplataforma (*mobile* e *desktop*) a partir de um único código fonte, este trabalho tem por objetivo estudar e prototipar uma solução de troca de mensagens utilizando o protocolo Simple (ou Streaming) Text Oriented Message Protocol (STOMP) e o intermediário de mensagens (*broker*) ApacheMQ.

2 FUNDAMENTAÇÃO TEÓRICA

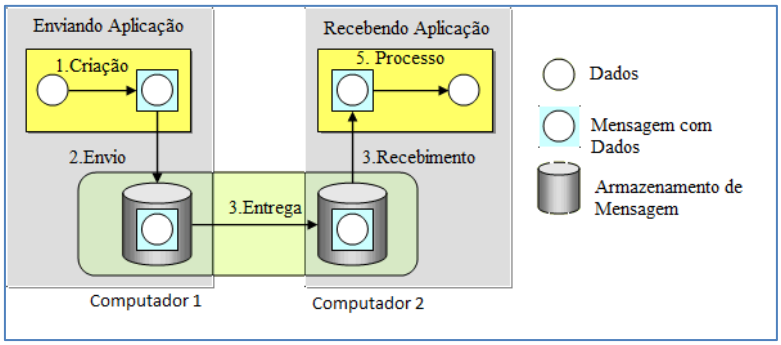
Esta seção apresenta definições e conceitos importantes, com a finalidade de fundamentação e aprofundamento acerca dos assuntos abordados neste trabalho e está organizada como segue: a seção 2.1 descreve os sistemas de mensageria e seu funcionamento; a seção 2.1.1 descreve o intermediador de mensagens (broker) e suas principais funcionalidades destacando uma tabela comparativa com as principais funcionalidades entre os intermediários de código aberto. A seção 2.1.2 descreve o funcionamento do protocolo STOMP. A seção 2.2 descreve sobre o servidor Apache ActiveMQ que será usado como base para o desenvolvimento deste trabalho. A seção 2.2.1 descreve sobre as principais características do servidor Apache. Por fim, a seção 2.3 traz um breve estudo sobre três trabalhos correlatos.

2.1 SISTEMAS DE MENSAGERIAS

Mensageria é uma tecnologia que permite comunicação de alta velocidade e de forma assíncrona entre sistemas diferentes com mensagens entregues de forma garantida (BASTOS, 2012). Um sistema de mensageria trabalha com esse conceito, coordenando e gerenciando as mensagens enviadas e recebidas. Seu objetivo é fazer com que nenhuma mensagem enviada seja perdida.

Para a transmissão de uma mensagem é utilizado o conceito de *sender*, *receiver* e canal, portanto o sender é quem envia a mensagem e escreve no canal. O receiver é quem recebe a mensagem e lê do canal, e os canais por sua vez são conhecidos como fila de mensagens. Os canais se comportam como se fossem uma coleção de mensagens compartilhadas entre múltiplos computadores que podem ser acessados por múltiplas aplicações (BASTOS, 2012). A primeira etapa para a transmissão de uma mensagem é a criação, quando o sender cria a mensagem preenchendo-a com os dados a serem enviados. A próxima etapa é a de envio, quando o sender adiciona a mensagem ao canal. O próximo passo é a entrega, quando o sistema de mensageria envia a mensagem do computador do sender, e deixa disponível ao computador do receiver. O receiver por sua vez, recebe e lê a mensagem no canal. Por fim, ocorre a etapa final que é o processamento da mensagem, quando o receiver extrai os dados da mensagem (Figura 5).

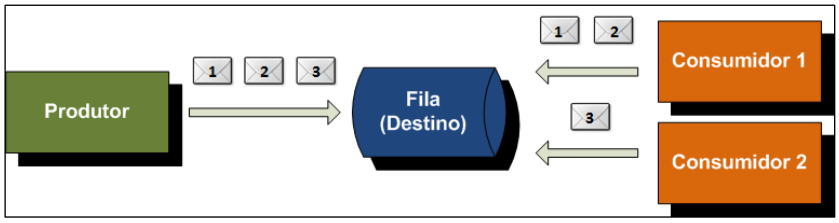
Figura 5 - Etapas para a transmissão de uma mensagem



Fonte: Baseado em Bastos (2012).

Os tipos de mensageria podem ser classificados em Point-to-Point(P2P) e Publish-and-Subscribe. O primeiro modelo utiliza como canal de comunicação uma fila na qual cada mensagem é enviada por um produtor a uma fila, onde ela permanece até que seja entregue a um consumidor. Esse modelo garante que a mensagem seja entregue a um único consumidor (destinatário). Por isso, mesmo que a fila possua mais que um consumidor ativo, apenas um receberá a mensagem (Figura 6).

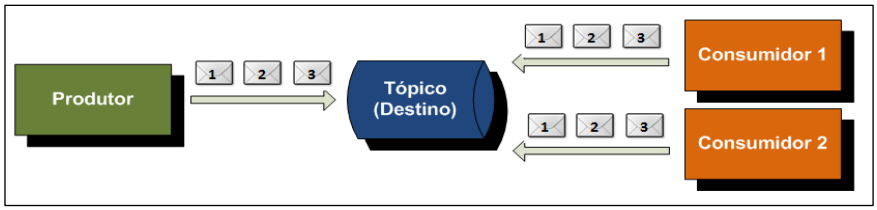
Figura 6 – Transmissão de mensagem point-to-point



Fonte: Souza (2014).

O modelo Publish-and-Subscribe também é conhecido como pub/sub. Esse modelo baseia-se no conceito de tópicos portanto, as mensagens são publicadas pelo produtor em um tópico (destino) e são entregues aos seus consumidores (aqueles que assinaram o tópico para recebimento de mensagens). Ao contrário do modelo Point-to-Point, este modelo permite que uma mesma mensagem seja entregue a vários consumidores (Figura 7).

Figura 7 – Transmissão de mensagens Publish-and-Subscribe



Fonte: Souza (2014).

2.1.1 INTERMEDIÁRIO DE MENSAGENS (BROKER)

Conforme Claro (2010), um intermediário de mensagens ou broker é um componente responsável por gerir a comunicação entre aplicações distintas. Neste contexto, as aplicações deixam de se comunicar diretamente o que evita a necessidade que uma aplicação precise conhecer a estrutura de outra. O uso de intermediário de mensagens tem a vantagem de permitir a disponibilização de diferentes interfaces de comunicação, podendo gerir o encaminhamento tendo em conta diversos aspectos que melhoram e asseguram a comunicação. Entre as funcionalidades, Claro (2010) cita:

a) clustering: envolve os processos de descoberta de serviços e intermediários, sistemas de *failover*, balanceamento de carga, qualidade de serviço, alta-disponibilidade, entre outros;
b) suporte para diversas linguagens e protocolos de mensagens: Para além da linguagem e protocolos de mensagens nativos suportados pelo intermediário, estes sistemas são implementados para comunicar através de protocolos como Web Services, AMQP e JMS que disponibilizam APIs para diversas linguagens;
c) suporte para diversos protocolos de transporte: A comunicação com o intermediário pode ser efetuada através de diferentes protocolos de transporte, tal como, in-VM para comunicações efetuados por aplicações em execução na mesma JVM do intermediário, UDP,TCP, NIO, SSL, *multicast*, entre outros;
d) persistência de mensagens: As mensagens recebidas podem ser persistidas, por exemplo em base de dados, para garantir que a mensagem nunca é perdida independentemente do consumidor estar indisponível ou de

existir alguma falha na entrega da mensagem. De forma a maximizar a eficiência e viabilidade do processo de persistência das mensagens são usadas caches e sistemas de log (*journaling*);

e) divisão de mensagens por destino: Podem ser configurados filtros e rotas de encaminhamento para que as mensagens possam ser replicadas ou desviadas;

f) Enterprise Application Integration: Através de padrões de integração existem mais protocolos e tecnologias de transporte suportadas facilitando o encaminhamento e transformação de mensagens;

g) ligação do intermediário para servidores com o mesmo protocolo: Permite configurar os intermediários de mensagens para que todas as mensagens recebidas para uma determinada fila ou tópico sejam encaminhadas para um servidor que suporte o mesmo protocolo. Por exemplo, se o intermediário suportar mensagens JMS pode ser configurada uma bridge com o serviço de mensagens JMS de alta performance do servidor aplicacional JBoss;

h) controle de fluxo: De forma a garantir que os limites de memória no intermediário não são excedidos, nas ligações síncronas é abrandado o fluxo dos produtores de mensagens atrasando o envio de confirmação da recepção da mensagem, enquanto em ligações assíncronas o produtor deve especificar um tamanho máximo para a janela de envio.

Claro (2010) realizou um levantamento sobre implementações de intermediários de mensagens e os classificou em dois grupos: com código aberto e proprietários. Entre os proprietários destacam-se IBM Websphere, TIBCO Enterprise Message Service Oracle WebLogic Server. Em código aberto existem soluções como ActiveMQ, HornetQ, OpenAMQ, OpenMQ, Qpid e RabbitMQ, cujas funcionalidades são apresentadas no Quadro 1.

Quadro 1- Comparação das principais funcionalidades de intermediários de mensagens de código aberto

	ActiveMQ	HornetQ	OpenAMQ	OpenMQ	Qpid	RabbitMQ
Linguagem de implementação	Java	Java	C	Java	Java e C++	Erlang
Protocolo de Mensagens	JMS 1.1	JMS 1.1	AMQP/0.9.1	JMS 1.1	AMQP/0.10	AMQP/0.8/0.9
Protocolos suportados	OpenWire, REST, Stomp,WSNotification, XMPP	Stomp, REST,Web sockets		Stomp		Stomp, XMPP
Linguagens suportadas	Java, Clientes Stomp, C/C++ (Stomp, OpenWire e CMS), Ajax (Rest), WebSockets, C# .NET (NMS), Delphy, FreePascal , (Habari) JavaScript (Ajax e WebSockets), etc.	Java, Clientes Stomp	C e aplicações de terceiros para Python, Java (JMS) e Ruby	Java e clientes Stomp	C++, Java (JMS), Python, Ruby, C#, .NET, Adaptor WCF e Ruby	C, Erlang e clients Stomp
Clustering	Descoberta, failover, balanceamento e replicação	Descoberta, failover, balanceamento e replicação	Descoberta failover, balanceamento (só de subscritores)	Descoberta, failover, balanceamento e replicação	Descoberta, fail over e replicação	Descoberta, failover, balanceamento e replicação
Protocolos relevantes	In-VM SSL (2-way)	In-VM SSL (2-way)		In-VM SSL (1-way)	In-VM SSL (2-way)	SSL (2-way)
Bridging	Sim	Sim	Não	Sim	Não	Não
Persistência	Sim	Sim	Não	Sim	Sim	Sim
Controle de fluxo	Sim	Sim	Não	Sim	Sim	Sim

Fonte: Claro (2010).

2.1.2 PROTOCOLO STOMP

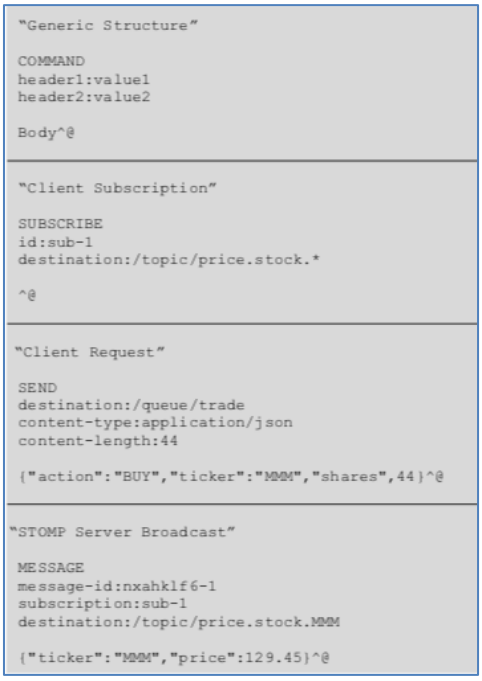
O STOMP é um protocolo que permite a comunicação assíncrona entre os clientes através de mensagens de um servidor mediador. Esse protocolo baseia-se em quadros, chamados frames que são basicamente: um comando (ou operação), uma mensagem (que seria o corpo) e um cabeçalho da mensagem. O STOMP é baseado em texto, mas também permite a transmissão de mensagens binárias. A codificação padrão é UTF-8, mas ele suporta a especificação de codificações alternativas para o corpo das mensagens (STOMP, 2012).

Conforme Danowski-Buhren (2016, p.57), através de mensagens de texto o protocolo STOMP trabalha com o conceito de Publish e Subscribe portanto, ele realiza um mecanismo de produtor/consumidor através de um terminal intermediário que gerencia os clientes conectados enviando qualquer mensagem recebida para um destino especificado. Um cliente que pretende transmitir uma mensagem para um grupo de clientes só pode enviar esta mensagem uma vez

para o terminal, e a partir daí a mensagem é transmitida para todos os clientes que estiverem conectados a ele. É possível também enviar uma mensagem para um único cliente desde que ele esteja conectado como um destino específico deste cliente. Para separar os destinos únicos dos destinos de grupo o protocolo STOMP recomenda o uso de dois prefixos /queue e /topic, onde topic é usado para destinos de grupo e queue para destinos únicos (STOMP, 2012).

A Figura 8 apresenta uma visão geral dos tipos de estruturas de mensagens mais importantes de acordo com as especificações STOMP (STOMP, 2012). No geral, uma mensagem é chamada de frame, e é formada por um comando, vários cabeçalhos e um corpo. Para cada uma das três ações principais uma estrutura (`SUBSCRIBE`, `SEND`, `RECEIVE`) STOMP é apresentada. Por exemplo, ao executar o frame `SUBSCRIBE`, os clientes entregam um id único e um destino ao qual pretendem se conectar. O intermediador de mensagens (broker) se encarrega de transmitir a mensagem para todos os clientes conectados ao destino especificado, identificando cada um deles pelo seu id único.

Figura 8 – Principais estruturas de mensagens STOMP



Fonte: Danowski-Buhren (2016, p.57).

O quadro começa com uma palavra chave terminada por uma nova linha. A palavra chave indica uma ação e pode ser, por exemplo, `CONNECT`, `SEND`, `SUBSCRIBE`, `UNSUBSCRIBE`, `ACK`, `NACK`, `BEGIN`, `COMMIT` e `DISCONNECT`. Após este comando há uma ou mais entradas de cabeçalho no formato`<key>: <valor>`. Cada entrada de cabeçalho é terminada por uma nova linha. Uma linha em branco indica o final dos cabeçalhos e do início do corpo da mensagem. O corpo é então seguido pelo byte nulo, representado neste exemplo por `(^ @)`. O cliente STOMP inicia o fluxo de conexão com o servidor enviando o comando `CONNECTED` no frame. Se o servidor aceitar a tentativa de conexão, ele irá responder no frame com o comando `CONNECTED`. Se o cliente e o servidor não compartilham todas as versões comuns de protocolo, então o servidor responde com um `ERRO`. Os comandos aceitos pelo protocolo são `SEND`, `SUBSCRIBE`, `UNSUBSCRIBE`, `BEGIN`, `COMMIT`, `ABORT`, `ACK`, `NACK` e `DISCONNECT`. Se o cliente enviar um outro comando diferente destes, o frame deve responder com um erro.

O comando `SEND` envia a mensagem para o seu servidor de destino. Ao utilizar este comando deve-se iniciar o cabeçalho `destination`, que indica para onde enviar a mensagem. O corpo do frame é a mensagem a ser enviada.

O comando `SUBSCRIBE` é utilizado para informar ou registrar que alguém está escutando em certo destino, com certo identificador. Há dois `headers:destination` e `id`. O `destination` é de onde deve-se ler/receber mensagens (fila), e o `id` é identificador da conexão (COLEN, 2012). Caso não seja possível fazer a conexão do cliente com o servidor, o servidor deve enviar ao cliente um erro, desconectando-o.

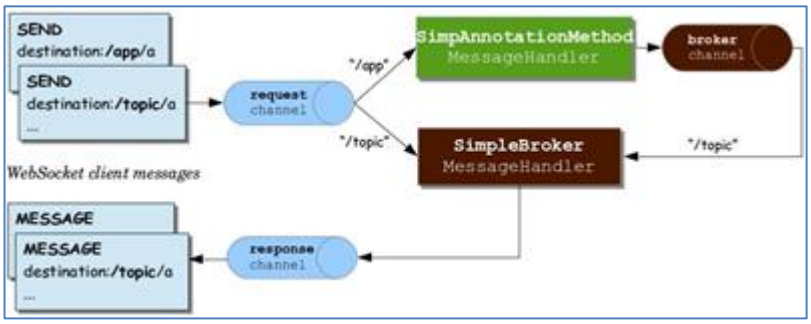
O comando `UNSUBSCRIBE` é usado para remover uma conexão existente. Uma vez removida a conexão o cliente não recebe mais as mensagens do seu destino. É obrigatório informar o cabeçalho `id` usado para o comando de uma conexão já feita.

O comando `DISCONNECT` permite que o cliente possa se desconectar do servidor a qualquer momento, porém não há nenhuma garantia de que os frames enviados anteriormente tenham sido recebidos pelo servidor. O cliente deve

enviar o comando `DISCONNECT`, e no cabeçalho o id `receipt` informando qual a conexão, e então aguardar a resposta `receipt` para desconectar.

De acordo com Danowski-Buhren (2016), através da configuração apropriada, um servidor de aplicação pode servir como uma extremidade de processamento em um ambiente STOMP, que recebe requisições de clientes prefixados com um destino configurado. O fluxo de trabalho do processamento de mensagens é demonstrado na Figura 9. O cliente recebe mensagens prefixadas com `/app`. Após ser realizado o processamento a resposta é enviada ao canal, que transmite a mensagem a todos os clientes conectados ao destino `/topic/a` (Danowski-Buhren (2016).

Figura 9 – Fluxo de processamento de mensagens



Fonte: Danowski-Buhren (2016, p.58).

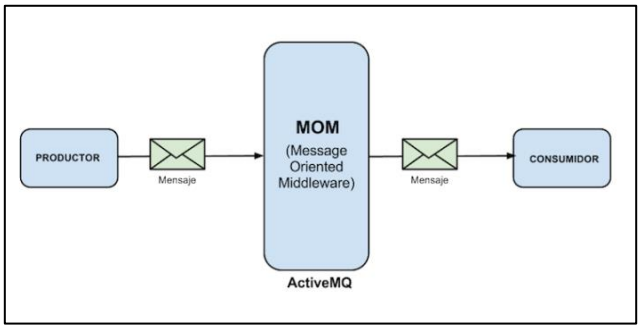
2.2 APACHE ACTIVEMQ

O Apache Active Message Queue é um projeto de código aberto de mensageria. Segundo Snyder, Bosanac e Davies (2011), ApacheMQ é uma solução de código aberto que implementa um middleware para troca de mensagens que permite a construção de aplicações de comunicação similares a WhatsApp utilizando o protocolo STOMP. Conforme STOMP (2019) há implementações de servidores do protocolo STOMP em várias linguagens, como: Erlang, Ruby, Java e Python. Por outro lado, há várias implementações de clientes em linguagens como: Perl, C++, C#, Java, C, Ruby, Objective C, Python, PHP, Erlang, Go, Javascript e TCL. A implementação em Delphi é mantida por Teti (2016).

Conforme Apache (2012), o Apache ActiveMQ é rápido, suporta várias linguagens e protocolos sendo responsável pela criação e gerenciamento de conexões de rede usados para a comunicação entre cliente e servidor. É um serviço orientado a Message Oriented Middleware (MOM) que oferece comunicação assíncrona entre cliente/servidor. É escrito em Java e permite a comunicação entre os sistemas que utilizam Java Message Service (JMS).

Quando o produtor enviar uma mensagem, ela é enviada ao serviço orientado a MOM, neste caso o servidor Apache ActiveMQ e ele se encarrega de enviar a mensagem diretamente ao consumidor. É ele o responsável por armazenar e realizar a troca de mensagens entre o produtor e o consumidor, conforme mostra a Figura 10.

Figura 10 – Estrutura geral de um Message Oriented Middleware



Fonte: Rodríguez (2019).

Um cliente JMS é uma aplicação Java que usa a API JMS para interagir com o provedor JMS, o ActiveMQ (URSO, 2013). Através dessa API, duas ou mais aplicações podem se comunicar por troca de mensagens. Dentre as principais características estão:

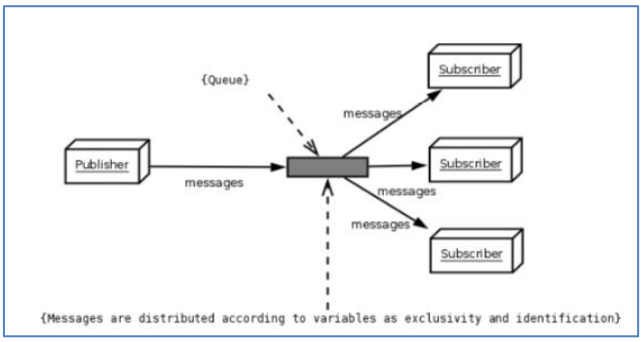
a) possuir código aberto e é distribuído sob a licença Apache;
b) atuar como mediador entre a troca de mensagens de envio e recebimento;
c) fornecer comunicação assíncrona entre aplicativos;
d) possuir suporte ao protocolo STOMP e a uma variedade de outros protocolos como HTTP;
e) suportar uma variedade de clientes e protocolos de Java, C, C++, C#, Ruby, Pyton, Php, etc;
f) é Open Wire para clientes de alto desempenho em Java, C, C ++, C #;
g) possuir interface de administração gráfica.

Conforme Pardo (2007,p.70), o padrão publish/ subscribe obtém escalabilidade e melhora o dinamismo da rede fazendo chamadas assíncronas para assinantes desconhecidos, podendo ou não responderem um ao outro dependente de sua declaração. Resumidamente, o produtor envia uma mensagem para um canal e esta mensagem é reenviada para todos os consumidores que estiverem conectados no canal, se houverem. Entretanto, quando as mensagens são importantes demais para serem perdidas, esse padrão precisa ser ampliado para definir novas necessidades, como:

a) processamento: as mensagens podem se perder sem ter um assinante para gerenciar os atributos da mensagem;
b) performance: o tráfego de rede é aumentado em um número n, sendo n o número de assinantes;
c) exclusividade: pode ser necessário que uma mensagem seja recebida apenas por um determinado assinante ou assinantes.

A solução Apache para os problemas anteriores era criar uma fila, e dentro dessa fila as mensagens são enviadas para os assinantes, podendo depender da velocidade do assinante, exclusividade do assinante ou do grupo de mensagens. Essa fila armazena todas as mensagens enviadas e entrega a mensagem somente quando um assinante se conecta, processa todas as mensagens, envia uma mensagem apenas uma vez, diminui o tráfego da rede e, se necessário, pode definir um mecanismo de seleção para escolher para qual assinante a mensagem deve ser enviada, ganhando exclusividade. Conforme Pardo (2007), a Figura 11 apresenta o mecanismo padrão de distribuição de filas do ActiveMQ.

Figura 11 – ActiceMQ queue: Publish Subscribe pattern application



Fonte: Pardo (2007,p.71).

2.3 TRABALHOS CORRELATOS

A seguir estão relacionados três trabalhos correlatos ao proposto. São eles: o aplicativo Gozirra, que é o item mais semelhante ao apresentado neste trabalho, conforme mostra o Quadro 2, um artigo que ensina a como criar um aplicativo de chat usando Spring Boot + WebSocket + RabbitMQ, conforme o Quadro 3 e a biblioteca STOMP.js, conforme o Quadro 4 com seus principais objetivos e funcionalidades.

Quadro 2 - Gozirra

Referência	Santos (2012)
Objetivos	Permitir troca de mensagens entre os usuários conectados
Principais funcionalidades	É uma implementação leve do protocolo Stomp. A partir do primeiro lançamento já inclui implementações de troca de mensagens entre cliente e servidor. É um aplicativo de bate papo do Android usando o protocolo Stomp com o servidor RabbitMQ.
Ferramentas de desenvolvimento	Protocolo STOMP, Java, Android e RabbitMQ
Resultados e conclusões	O trabalho atende o objetivo pretendido e foram liberadas novas versões de correções a partir da versão inicial.

Fonte: Elaborado pelo autor.

Quadro 3 – Tutorial para criar um aplicativo de chat usando Spring Boot + WebSocket + RabbitMQ

Referência	Maniyar (2019)
Objetivos	Permitir troca de mensagens entre vários usuários em tempo real
Principais funcionalidades	A sua principal funcionalidade é permitir que um usuário logado pode enviar e receber mensagens diretamente do servidor RabbitMQ em tempo real.
Ferramentas de desenvolvimento	Spring Boot, WebSocket, RabbitMQ
Resultados e conclusões	É possível verificar que ao final do tutorial é possível a troca de mensagens entre cliente e servidor simultaneamente em tempo real.

Fonte: Elaborado pelo autor.

Quadro 4 – Biblioteca STOMP.js

Referência	Mesnil (2015)
Objetivos	Permitir aos usuários utilizarem esta API para envio e recebimento de mensagens
Principais funcionalidades	Esta biblioteca fornece um cliente STOMP para o navegador da Web (usando Web Sockets) ou aplicativos node.js. O projeto contém exemplos para o uso de stomp.js para envio e recebimento de mensagens STOMP de um servidor diretamente no Navegador da Web ou em um WebWorker.
Ferramentas de desenvolvimento	STOMP, Web Sockets
Resultados e conclusões	A biblioteca disponibilizada pelo projeto foi concluída e apresenta exemplos de uso para enviar e receber mensagens. O projeto apresenta página com toda a documentação e um projeto de teste para executar no browser.

Fonte: Elaborado pelo autor.

O projeto de MESNIL (2015) serviu como base de estudo para o desenvolvimento do protótipo estudado neste trabalho. A biblioteca disponibilizada foi totalmente desenvolvida em cima do protocolo STOMP, alvo de estudo para o desenvolvimento deste trabalho. A partir dos exemplos apresentados pelo projeto foi possível testar a usabilidade e o funcionamento da aplicação. Já o projeto de Santos (2012) é uma aplicação desenvolvida com o protocolo STOMP porém com o servidor RabbitMQ. A partir dele foi possível aprofundar o estudo no protocolo e estudar suas principais funções. O projeto de MANIYAR (2019) permite a troca de mensagens entre os usuários em tempo real utilizando Web Sockets e o servidor RabbitMQ. Através dele foi possível estudar de forma aprofundada as funções entre cliente e servidor em tempo real.

3 DESENVOLVIMENTO DO PROTÓTIPO

A seguir são apresentados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF), as etapas e a metodologia de desenvolvimento.

3.1 REQUISITOS

No Quadro 5 são descritos os Requisitos Funcionais (RF) e os respectivos Casos de Uso (UC).

Quadro 5 – Requisitos Funcionais

Requisito Funcional (RF)	Caso de Uso (UC)
RF01: permitir realizar o login do usuário	UC - 001
RF02: permitir manter usuários	UC - 002
RF03: permitir cadastrar um grupo	UC - 004
RF04: permitir manter grupos	UC - 003
RF05: permitir a troca de mensagens entre dois usuários em conversa privada	UC - 006
RF06: permitir a troca de mensagens entre três usuários ou mais em conversa de grupo	UC – 007
RF07: permitir o envio de imagem para um ou mais usuários	UC - 008
RF08: permitir que o usuário consulte o histórico das conversas privadas	UC - 009

Fonte: Elaborado pelo autor.

No Quadro 6 são descritos os Requisitos Não Funcionais (RNF).

Quadro 6 – Requisitos Não Funcionais

Requisito Não Funcional (RNF)
RNF01: o aplicativo deve ser desenvolvido na linguagem de programação Delphi
RNF02: utilizar o servidor Apache ActiveMQ
RNF03: utilizar o protocolo STOMP para integração com o aplicativo

Fonte: Elaborado pelo autor.

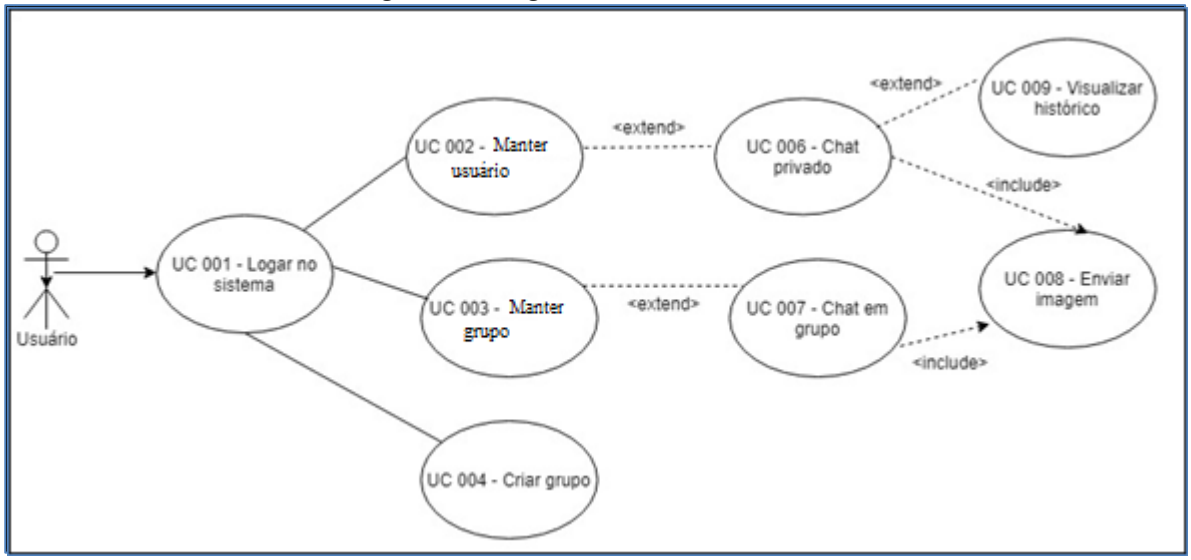
3.2 ESPECIFICAÇÃO

A especificação da aplicação foi criada utilizando a ferramenta Draw.io. Com esta ferramenta foram desenvolvidos os diagramas de casos de uso e de atividades da Unified Modeling Language (UML).

3.2.1 Diagrama de Caso de Uso

O diagrama apresentado na Figura 12 descreve os casos de uso da aplicação. Foi identificado um único papel, que é o de usuário, aquele que vai utilizar o sistema. O caso de uso `UC 001 – Logar no sistema` é utilizado para o usuário fazer o login na aplicação e utilizar suas funcionalidades. Nesse momento o usuário fica online, disponível para entrar em alguma conversa (chat). A partir desse momento o usuário pode escolher entre selecionar um usuário na lista de usuários cadastrados na aplicação, conforme o caso de uso UC `002 – Manter usuário`, ou pode selecionar um grupo entre os grupos já cadastrados na aplicação, conforme o caso de uso `UC 003 – Manter grupo` ou cadastrar um novo grupo, conforme o caso de uso `UC 004 – Criar grupo`.

Figura 12 – Diagrama de Caso de Uso



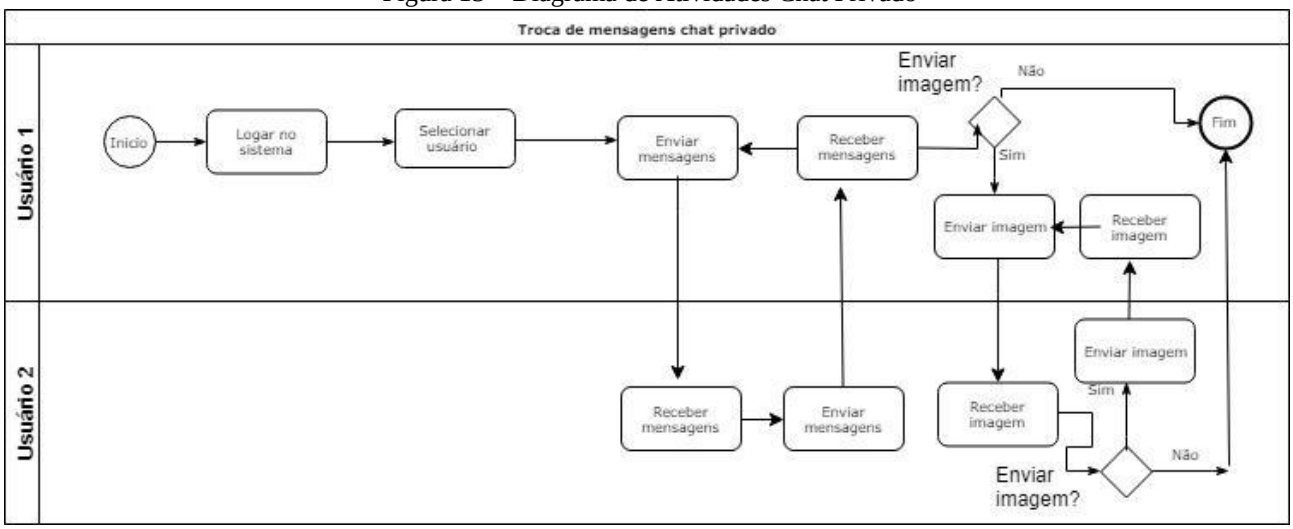
Fonte: Elaborado pelo autor.

O caso de uso `UC 006 – Chat privado` permite que o usuário inicie uma conversa privada com outro usuário selecionado conforme o caso de uso `UC 002 – Selecionar usuário`. Já o caso de uso `UC 007 – Chat em grupo`, permite que o usuário inicie uma conversa em grupo com outros usuários conectados a este mesmo grupo, conforme o grupo selecionado no caso de uso `UC 003 – Selecionar grupo`. O caso de uso `UC 008 – Enviar imagem` permite ao usuário enviar uma imagem por vez para um usuário no chat privado ou, para vários usuários no chat em grupo. Por fim, o caso de uso `UC 009 – Visualizar histórico` permite que o usuário logado tenha acesso ao seu histórico de conversas com outros usuários.

3.2.2 Diagrama de Atividades

A Figura 13 mostra o diagrama de atividades de uma simulação de chat em conversa privada.

Figura 13 – Diagrama de Atividades Chat Privado

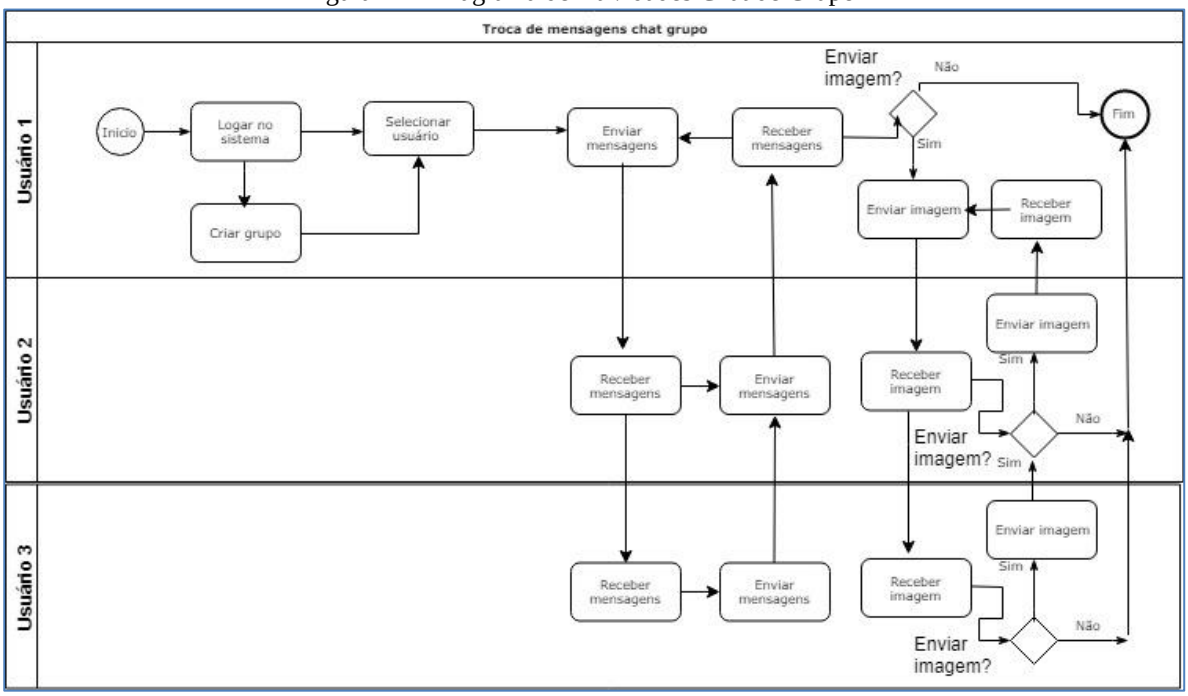


Fonte: Elaborado pelo autor.

O `Usuário 1` faz seu login no sistema. Nesse momento se o usuário nunca acessou o sistema, ele é criado no servidor, caso contrário é realizado seu login. Após autenticar-se, o usuário 1 seleciona um outro usuário a partir de uma lista de usuários cadastrados no sistema para iniciar uma conversa, e então envia mensagens para o usuário 2. O `Usuário 2` recebe as mensagens enviadas para ele e envia novas mensagens para o usuário 1. O usuário 1 recebe as mensagens enviadas pelo usuário 2 e pode escolher se deseja enviar alguma imagem. Caso positivo, ele envia uma imagem para o usuário 2. O usuário 2 recebe a imagem enviada e pode também enviar uma imagem para o usuário 1. Resumidamente, os usuários podem trocar mensagens e imagens simultaneamente.

A Figura 14 mostra o diagrama de atividades de uma simulação de chat em conversa de grupo.

Figura 14 – Diagrama de Atividades Chat de Grupo



Fonte: Elaborado pelo autor.

Após autenticar-se no sistema, o `Usuário 1` tem a opção de selecionar um grupo a partir de uma lista de grupos cadastrados no sistema para iniciar uma conversa, ou então criar um grupo e a partir dele então enviar mensagens para todos os usuários incluídos neste grupo. Neste diagrama foi colocado como exemplo um grupo de três usuários. O `Usuário 2` e o `Usuário 3` recebem as mensagens enviadas para o grupo e enviam novas mensagens para o dentro do mesmo grupo, fazendo com os outros usuários conectados a este grupo recebam as mensagens. Qualquer um dos usuários conectados ao grupo pode enviar uma imagem. O usuário 1 envia uma imagem para o grupo e o usuário 2 e o usuário 3 recebem a imagem, estando eles conectados a este grupo. Resumidamente, este diagrama é semelhante ao anterior, porém permite que mais de dois usuários troquem mensagens ou imagens, apresentando o conceito de grupo dentro da aplicação.

3.3 ESTRUTURA DO PROJETO

Inicialmente o esforço da pesquisa foi para encontrar uma biblioteca em Delphi que trabalhasse com o protocolo STOMP. Nesta etapa foi localizado no github o projeto Delphi STOMP Client de TETI (2016). Para criação deste trabalho foi usado a biblioteca StompClient.pas do trabalho citado. A biblioteca permite implementar todas as chamadas ao servidor ApacheMQ que atua como servidor de backend do projeto.

3.4 ARQUITETURA

Esta seção descreve a camada de front-end (o aplicativo de mensagens) que é configurada para conectar-se com o servidor ApacheMQ (que atua como back-end).

3.4.1 FRONT-END

Essa camada é responsável por garantir a interação do usuário, permitindo autenticar-se na aplicação e escolher entre criar uma conversa privada, portanto somente dois usuários trocam mensagens entre si ou criar uma conversa de grupo, permitindo trocar mensagens entre dois usuários ou mais ao mesmo tempo. A aplicação permite também o envio de imagens de um usuário ao seu destinatário sendo ela uma conversa privada ou imagens de um usuário para vários usuários de um grupo, permitindo enviar uma imagem por vez. Para ampliar a visualização da imagem recebida basta

posicionar o mouse sobre a imagem e ela é apresentada em zoom maior. A seguir é descrito de forma detalhada todas as funcionalidades que o sistema oferece.

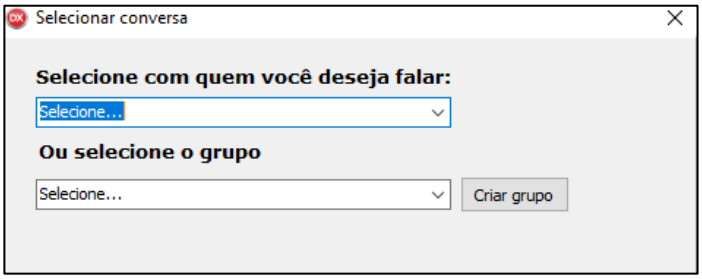
Para ter acesso a aplicação é necessário realizar o processo de login. Para isto, basta informar um usuário e senha de acesso e clicar no botão `Entrar`, conforme demonstra a Figura 15. Cada usuário é cadastrado como um novo topic do servidor ApacheMQ então, se o usuário já existir vai entrar diretamente na conta dele. Caso contrário, a aplicação vai criar um usuário novo na lista de topics.

Figura 15 – Tela de login inicial da aplicação

Fonte: Elaborado pelo autor.

Ao realizar o login na aplicação é apresentada a tela sala global. Nesta tela o usuário pode escolher se deseja selecionar um outro usuário para iniciar uma conversa privada ou se deseja selecionar um grupo de pessoas para conversar, conforme mostra a Figura 16.

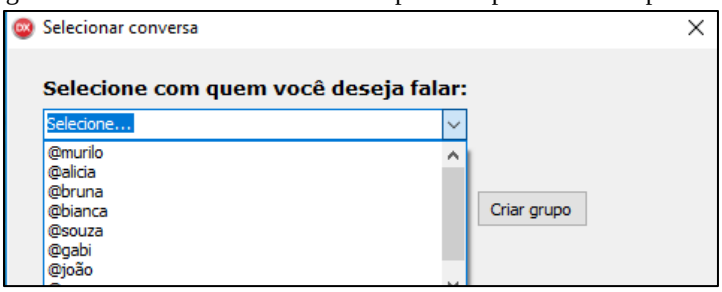
Figura 16 – Tela de sala global



Fonte: Elaborado pelo autor.

Todos os usuários que efetuaram o login pelo menos uma vez na aplicação já estão cadastrados e disponíveis para iniciar uma conversa. Ao selecionar o primeiro combo box todos esses usuários são listados conforme mostra a Figura 17.

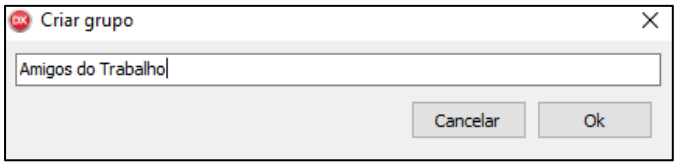
Figura 17 – Usuários cadastrados e disponíveis para conversa privada



Fonte: Elaborado pelo autor.

Para realizar uma conversa de grupo existem duas opções que o usuário pode selecionar: criar um novo grupo ou selecionar um grupo já cadastrado para iniciar uma conversa. Para criar um novo grupo basta clicar no botão `Criar grupo` e informar um nome para ele, conforme mostra a Figura 18.

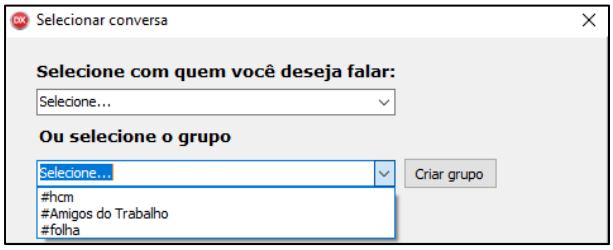
Figura 18 – Criando um novo grupo



Fonte: Elaborado pelo autor.

Após isso, o nome do grupo criado já é apresentado no segundo combo box da sala global, assim como todos os grupos já cadastrados na aplicação, permitindo a inicialização de uma conversa em grupo, conforme mostra a Figura 19.

Figura 19 – Grupos cadastrados e disponíveis para conversa de grupo

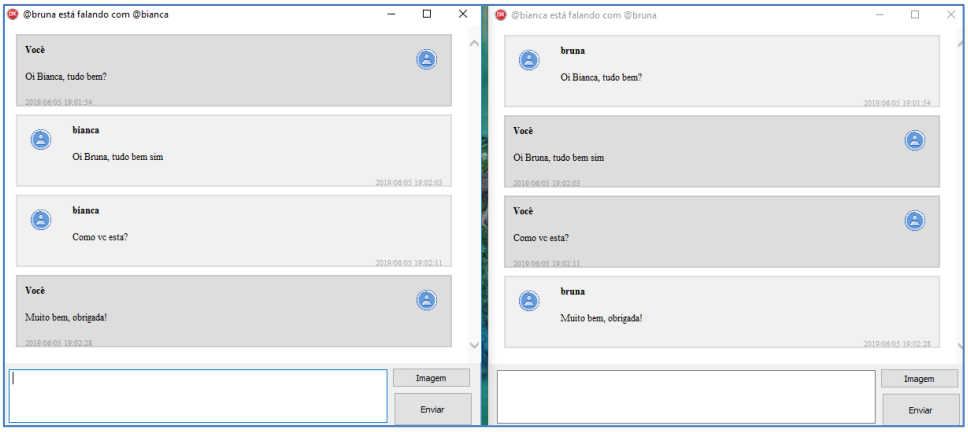


Fonte: Elaborado pelo autor.

3.4.1.1 CHAT PRIVADO

Ao selecionar uma pessoa na lista de usuários cadastrados é apresentada a tela do chat. Caso o destinatário esteja online na aplicação as mensagens enviadas são entregues imediatamente a ele, sendo que a mensagem enviada pelo remetente pode ser recebida por um único destinatário por vez, criando uma relação de um pra um. Para enviar uma mensagem basta o remetente preencher a caixa de diálogo e clicar no botão `Enviar`. A Figura 20 apresenta um diálogo em um chat privado. A esquerda o usuário `bruna` está iniciando um diálogo com o usuário `bianca`, que está a direita.

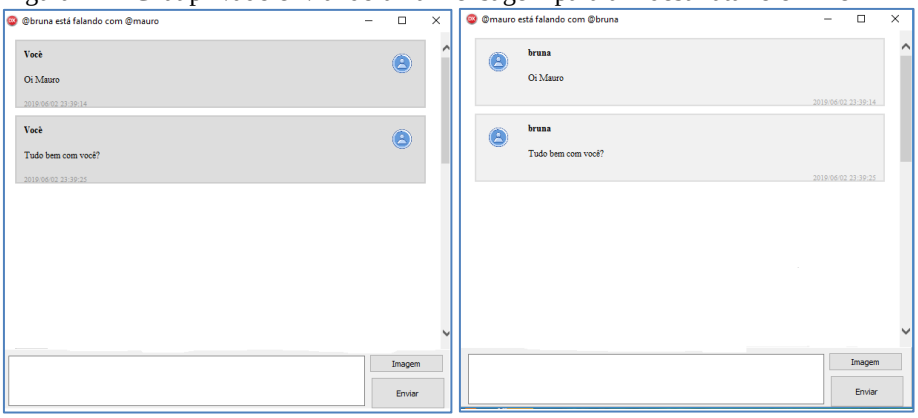
Figura 20 – Chat privado enviando uma mensagem para um destinatário online



Fonte: Elaborado pelo autor.

Caso o usuário destinatário não esteja online no momento que outro usuário enviar uma mensagem para ele, as mensagens enviadas ficam armazenadas como histórico e assim que o usuário acessar a aplicação e selecionar o usuário que enviou a mensagem para iniciar uma conversa elas serão apresentadas na tela da aplicação, conforme mostra a Figura 21. As mensagens enviadas pelo remetente são destacadas em tom cinza escuro e as mensagens do destinatário são destacadas em tom cinza claro.

Figura 21 – Chat privado enviando uma mensagem para um destinatário offline

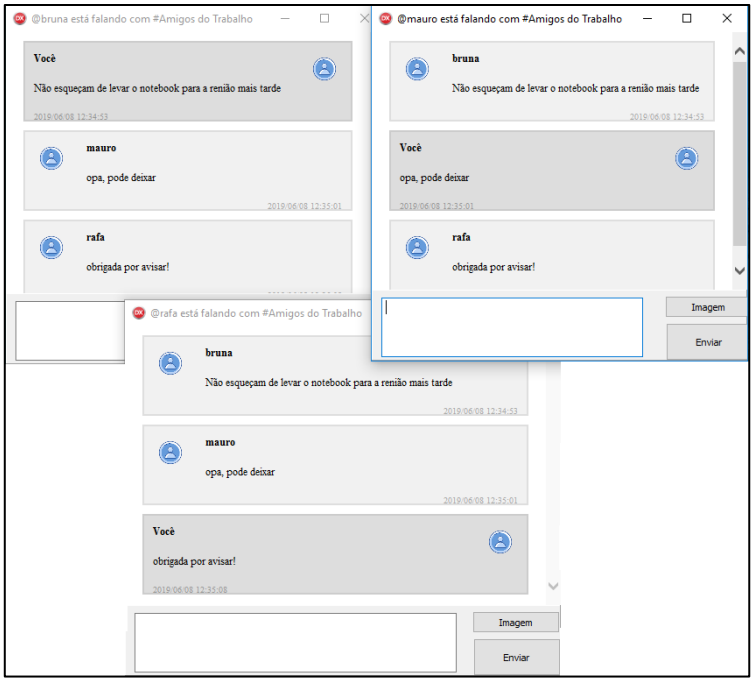


Fonte: Elaborado pelo autor.

3.4.1.2 CHAT EM GRUPO

Ao selecionar um grupo na lista de grupos cadastrados é apresentada a tela do chat. Diferente do chat privado, o chat de grupo não armazena um histórico das conversas, ou seja, só recebe as mensagens enviadas aqueles usuários que estiverem logados no grupo selecionado no momento em que o remetente enviou a mensagem, sendo que a mensagem enviada pode ser recebida por um ou mais usuários ao mesmo tempo. Para enviar uma mensagem para um grupo basta preencher o remetente preencher a caixa de diálogo e clicar no botão `Enviar`. A Figura 22 apresenta um diálogo em um chat de grupo com três usuários on-line. A esquerda o usuário `bruna`, a direita o usuário `mauro` e abaixo o usuário `rafa`.

Figura 22 – Demonstração de conversa em grupo

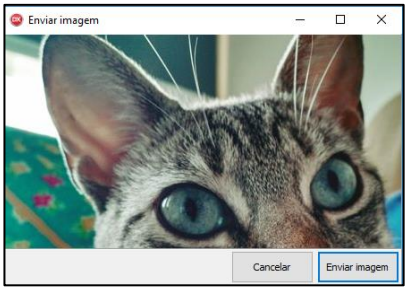


Fonte: Elaborado pelo autor.

3.4.1.3 ENVIO DE IMAGEM

Tanto o chat privado quanto o chat de grupo permitem o envio de imagens para o destinatário sendo possível enviar uma imagem por vez para um ou mais destinatários. Para enviar uma imagem basta o remetente clicar no botão `Imagem` e ao selecionar a imagem desejada ela é carregada e demonstrada em uma tela para que o usuário visualize a imagem antes de enviá-la, permitindo escolher enviar ou cancelar o envio da imagem, conforme mostra a Figura 23.

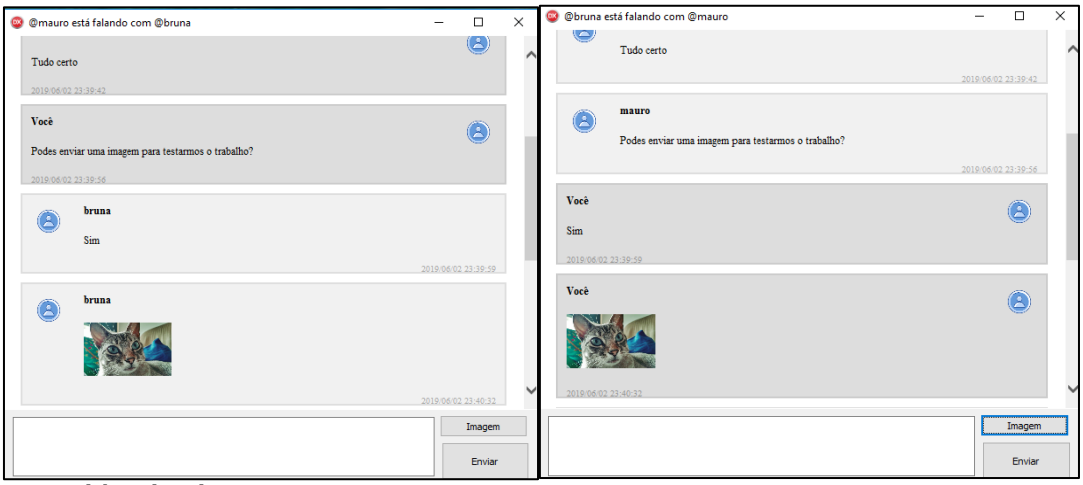
Figura 23 - Tela de visualização de imagem



Fonte: Elaborado pelo autor.

Ao clicar no botão `Enviar` a imagem é enviada a um único usuário caso a conversa seja privada ou para todas as pessoas conectadas ao grupo selecionado, sendo demonstrada em forma de miniatura na tela do chat, conforme mostra a Figura 24.

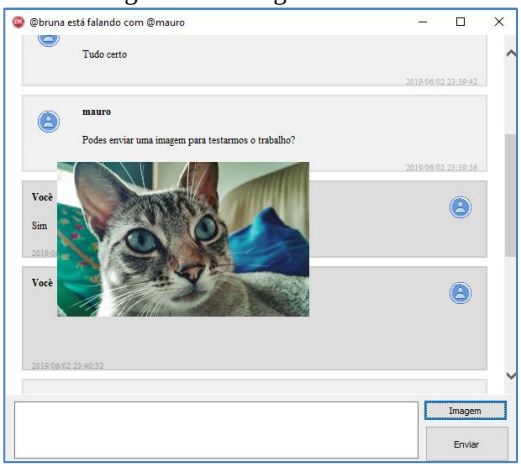
Figura 24 - Envio de imagem para destinatário único



Fonte: Elaborado pelo autor

Para visualizar a imagem em tamanho maior basta que o usuário arraste o mouse e posicione sobre a imagem. A imagem será então apresentada em tamanho maior no formato de zoom, conforme mostra a Figura 25. Para voltar a visualizar a imagem em miniatura novamente basta arrastar o mouse para fora da imagem.

Figura 25 – Imagem com zoom



Fonte: Elaborado pelo autor.

3.4.2 BACK-END

O Back-end é a camada responsável por realizar o login do usuário na aplicação e permitir a troca de mensagens e imagens entre os usuários conectados. O servidor responsável por armazenar estes usuários e suas respectivas mensagens é o Apache ActiveMQ.

4 CONCLUSÕES

Este trabalho descreveu as etapas de construção de um protótipo de aplicação que permite a troca de mensagens entre os usuários conectados utilizando o protocolo Stomp, o ambiente de desenvolvimento Delphi e o servidor Apache ActiveMQ.

Como resultados obtidos, pode-se verificar que foi possível construir um aplicativo que permite realizar a troca de mensagens entre usuários conectados individualmente ou em grupo, permitindo o cadastro de usuários e grupos na aplicação. As conversas privadas são armazenadas como histórico, portanto, se o destinatário não estiver online no momento em que a mensagem for enviada a ele, quando este usuário se conectar a aplicação para conversar com o usuário remetente todas as mensagens que foram enviadas a ele serão apresentadas na tela. A aplicação permite também a troca de imagens entre os usuários conectados podendo enviar uma imagem por vez para o/os destinatário(s). A implementação deste protótipo para testes de usuário final, bem como seu código fonte podem ser encontrados em: https://github.com/bgessner1607/TCC-Chat-Delphi-Stomp-ApacheMQ.

O trabalho demandou um esforço de pesquisa e testes em relação ao funcionamento da aplicação para plataforma Android, porém sem sucesso o que invalidou a proposta original de verificar se seria possível o desenvolvimento de uma aplicação multiplataforma utilizando a biblioteca backend Stomp com o ambiente de desenvolvimento Delphi. Durante os estudos aplicados na geração de código para Android foi visto que a biblioteca *StompClient.pas* utilizada como backend deste trabalho não permitiu gerar código para a linguagem devido a forma como ela foi construída pelo autor. Foi verificado que a biblioteca não está disponível no FMX (*Firemonkey*), plataforma Delphi para a linguagem Android. O trabalho possibilitou a acadêmica ampliar os conhecimentos adquiridos durante a graduação e aprofundar seus conhecimentos em termos de protocolo de comunicação e comunicação com servidores de mensagens.

Como extensões ao trabalho propõe-se:

a) validação se é possível a comunicação entre clientes desenvolvidos em Delphi e servidores de mensagens como ApacheMQ utilizando-se websockets;

b) desenvolver um conjunto de dados de testes de carga para verificar a capacidade de atendimento do servidor e do cliente sob altas cargas de mensagens;

c) verificar a possiblidade de integração de clientes de comunicação desenvolvidos em Delphi com outros servidores de mensagens e utilizando outros protocolos de comunicação;

d) estudar como garantir privacidade das comunicações utilizando clientes stomp e serviços de mensageria.

REFERÊNCIAS

APACHE.**ActiveMQ**. [S.l.], [2012]. Disponível em: < http://activemq.apache.org/>. Acesso em: 04 abr. 2016.

BARROS, Thiago. **Baixe Hangouts: a convergência dos serviços de mensagem do Google**. [S.l.], [2014]. Disponível em: < http://www.techtudo.com.br/tudo-sobre/hangouts.html/>. Acesso em: 28 mar. 2016.

BASTOS, Lucas. **Introdução a mensageria**. [S.l.], [2012]. Disponível em: <https://www.concrete.com.br/2012/01/23/introducao-a-mensageria//>. Acesso em: 29 mar. 2016.

BRAZAN, Marina. **A era dos aplicativos mobile**. [S.l.], [2012]. Disponível em: < https://www.mirago.com.br/a-era-dos-aplicativos-mobile/>. Acessoem: 04 abr. 2016.

BUHREN, Christian Danowski. **Synchronizing Web-based 3D Information Visualization and Heterogeneous Data Sources.** 2016.95 f. Dissertação (Mestrado), University of Hagen, 2016. Disponível em: < https://www.researchgate.net/publication/309010786_Synchronizing_Webbased_3D_Information_Visualization_and_ Heterogeneous_Data_Sources >. Acesso em: 28 jun. 2019.

CLARO, Luís Miguel Chaves. **Arquitectura Orientada a Eventos**. 2010. 95 f. Dissertação (Mestrado) - Curso de Engenharia Informática, Escola de Engenharia, Universidade do Minho, Minho, 2010. Disponível em: <https://mei.di.uminho.pt/sites/default/files/dissertacoes//eeum_di_dissertacao_pg12817.pdf>. Acesso em: 10 mar. 2019.

COLEN, Thiago.**Um pouco de Stomp**. [S.l.],[2012]. Disponível em: <http://thiagocolen.blogspot.com.br/2012/09/um-pouco-de-stomp.html />. Acesso em: 28 mar. 2016.

FOX, Tim. **JBossJMSNewPerformanceBenchmark**. 2006. Disponível em: <https://developer.jboss.org/wiki/JBossJMSNewPerformanceBenchmark>. Acesso em: 17 jun. 2019.

HORNETQ, Red Hat. Inc. Jboss. **HornetQ - putting the buzz in messaging.** 2019. Disponível em: <www.jboss.org/hornetq>. Acesso em: 17 jun. 2019.

ICLARIFIED. **Skype App Gets Improved Dialer, Lets You Quickly Make Calls From the New Chat Picker.** 2015 .Disponivel em: < http://www.iclarified.com/46561/skype-app-gets-improved-dialer-lets-you-quickly-make-calls-from-the-new-chat-picker>. Acesso em 28 mar. 2016.

MANIYAR, Vijay. **Crie um aplicativo de chat usando Spring Boot + WebSocket + RabbitMQ**. 2019. Disponível em: <https://dzone.com/articles/build-a-chat-application-using-spring-boot-websock>. Acesso em: 28 jun. 2019.

MESNIL, Jeff. **STOMP.js**. 2015. Disponível em: <git://github.com/jmesnil/stomp-websocket.git>. Acesso em: 28 jun. 2019.

OPENAMQ. **OpenAMQ Enterprise AMQP Messaging.** 2019. Disponível em: <http://openamq.wikidot.com/>. Acesso em: 17 jun. 2019.

OPENMQ. **Open MessageQueue (Open MQ) -- A complete JMS MOM Platform.** 2019. Disponível em: <https://javaee.github.io/openmq/>. Acesso em: 17 jun. 2019.

PACETE, Luiz Gustavo. **Os potenciais inexplorados da mensageria**. 2019. Disponível em: <https://www.meioemensagem.com.br/home/marketing/2019/01/24/os-potenciais-inexplorados-da-mensageria.html>. Acesso em: 25 mar. 2019.

PAIVA, Fernando (Org.). **Panorama Mobile Time/Opinion Box:** Mensageria no Brasil. 2019. Disponível em: <https://panoramamobiletime.com.br>. Acesso em: 29 mar. 2019

PARDO, Diego Fernando Rivera. **On aspect-oriented concurrent and distributed patterns.** 2007. Disponível em:<http://web.imt-atlantique.fr/x-info/emoose/alumni/thesis/drivera.pdf >

POYARES, Walter. **Comunicação Social e Relações Públicas**. Rio de Janeiro: Agir, 1970.

QPID, Apache. Apache Qpid Messaging built on AMQP. 2019. Disponível em: <https://qpid.apache.org/>. Acesso em: 17 jun. 2019.

RABBITMQ. **RabbitMQ is the most widely deployed open source message broker.** 2019. Disponível em: <http://www.rabbitmq.com>. Acesso em: 17 jun. 2019.

REIS, Daniel. Estudo de uma plataforma aberta para comunicação M2M. Mestrado Integrado em Engenharia Informática e Computação. Faculdade de Engenharia da Universidade do Porto. 2012.

RODRÍGEZ, Miguel. Introdução ao Apache ActiveMQ.2019. Disponível em: <https://www.adictosaltrabajo.com/author/miguelarlandy/>. Acesso em: 13jun. 2019.

SANTOS, Anderson. **Gozirra**. 2012. Disponível em: <https://github.com/asantos2000/RabbitMQGozirraStompAndroid>. Acesso em: 28 jun. 2019.

SOUZA, Luiz. **JMS API: comunicação Assíncrona – Parte 1**. [S.l.], [2014]. Disponível em: < http://www.devmedia.com.br/jms-api-comunicacao-assincrona-parte-1/30073/>. Acesso em: 28 mar. 2016.

SNYDER, Bruce; BOSANAC, Dejan; DAVIES, Rob. **ActiveMQ in action.** Manning Publications Co..2011. 406p.

STOMP: **STOMP Implementations**. Version 1.2. [S.l.][2012], Disponível em: <https://stomp.github.io/implementations.html>. Acesso em: 15 jun. 2019.

STOMP.**STOMP Protocol Specification**: platform support. Version 1.1. [S.l.], [2012]. Disponível em: <https://stomp.github.io/stomp-specification-1.1.html#Protocol_Overview />. Acesso em: 28 mar. 2016.

TEIXEIRA, Marcelo. **Da comunicação humana a comunicação em rede: uma pluralidade de convergências.** 2012. Disponível em: <http://www.insite.pro.br/2012/fevereiro/comunicacao_redes_convergencias.pdf>. Acesso em: 28 mar. 2016.

TETI, Daniele. **Delphi/FreePascal STOMP Client**. 2016. Disponível em: <https://github.com/danieleteti/delphistompclient>. Acesso em: 5 mar. 2019.

URSO, Giuseppe. **JMS ActiveMQandthefailoverprotocol**. [S.l.], [2013]. Disponível em: < http://www.giuseppeurso.eu/en/jms-activemq-and-the-failover-protocol//>. Acesso em: 28 mar. 2016.

WHATSAPP.**WhatsApp**. [S.l.], [2016]. Disponível em: < https://www.whatsapp.com/>. Acesso em: 04 abr. 2016.