ARQUITETURA DE MICROSERVIÇOS PARA PERCEPÇÃO DA COLABORAÇÃO EM UM SISTEMA DE CHAT

Matheus Losi, Luciana Pereira de Araújo Kohler – Orientadora

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

mlosi@furb.br, lpa@furb.br

Resumo: *Este artigo apresenta uma aplicação de chat com foco em aspectos de percepção para a colaboração. A aplicação foi desenvolvida em uma arquitetura de microserviços, sendo que os serviços de backend foram desenvolvidos com a linguagem Python e a aplicação de frontend foi desenvolvida em Angular. O diferencial da aplicação são os aspectos de percepção no frontend, além da componentização dos serviços de backend utilizando Docker, fazendo com que o funcionamento dos serviços seja independente, assim, facilitando a manutenção e escalabilidade dos mesmos.*

Palavras-chave*: Microserviços. Python. Docker. Percepção. Chat.*

1 INTRODUÇÃO

Nos últimos anos a arquitetura de microserviços vem se consolidando cada vez mais no mercado, tornando-se comum no desenvolvimento de aplicações empresariais (LEWIS; FOWLER, 2014). Isso ocorre, pois as aplicações desenvolvidas na arquitetura monolítica ficaram complexas e muito grandes, dificultando a manutenção e a divisão do trabalho dentro das equipes de tecnologia. Um exemplo disso, são os ciclos de mudanças que são mais complexos que, segundo Lewis e Fowler (2014):

> Uma pequena alteração feita em uma parte pequena do software faz com que toda a aplicação monolítica necessite ser republicada. Com o passar do tempo ficará cada vez mais difícil manter uma estrutura modular, sendo difícil separar as mudanças que deveriam afetar somente um módulo (LEWIS; FOWLER, 2014, p. 1).

Nesse sentido, a arquitetura de microserviços se baseia na construção de serviços totalmente independentes com uma carga menor de responsabilidades, possibilitando assim o desenvolvimento de uma aplicação mais simples de fornecer manutenção e desenvolver novas funcionalidades (LEWIS; FOWLER, 2014). Microserviços são pequenos serviços autônomos e trabalham em conjunto (NEWMAN, 2015). Além disso, com esta arquitetura é possível a escolha de diferentes linguagens e estruturas de armazenamento de dados para uma mesma aplicação final, tendo em vista que cada microserviço pode ter sua própria linguagem e forma de gerenciamento de dados. Em geral, essa arquitetura utiliza a comunicação por Interface de Programação de Aplicações (API) no modelo de Transferência de Estado Representacional (REST) através de requisições com o Protocolo de Transferência de Hipertexto (HTTP).

Usar uma arquitetura de microserviços na construção de sistemas colaborativos pode trazer benefícios como a facilidade de liberar novos módulos e a facilidade de controlar funcionalidades diferentes para cada aplicação. Por exemplo, uma aplicação que utiliza o microserviço de mensagens, não necessariamente precisa utilizar o microserviço de compartilhamento de mídias.

Relacionado a sistemas colaborativos, um aspecto fundamental é a percepção (PIMENTEL; FUKS, 2011). Ela está relacionada a um usuário entender as ações que foram executadas por ele, bem como pelos outros usuários do mesmo grupo, evitando assim o isolamento e um melhor entendimento em ambientes de trabalho distribuídos. Também existem meios computacionais de dar auxílio a percepção de um usuário, notificando e usando artefatos do ambiente compartilhado para contextualiza-lo ao ambiente (PIMENTEL; FUKS, 2011, p.157).

Como a percepção, o contexto também tem um papel importante nos sistemas colaborativos, pois auxilia na compreensão, resolução de problemas ou aprendizagem (PIMENTEL; FUKS, 2011, p.160). O contexto é o agrupamento de condições relevantes que facilitam a compreensão do usuário a uma determinada situação, ações ou eventos.

Diante desse cenário, este trabalho apresenta o desenvolvimento de uma arquitetura de microserviços voltada para atender as características de percepção e contexto de um sistema colaborativo, a ser validada em uma aplicação de chat. A aplicação de chat foi escolhida para a validação, pois possui vários dos recursos de percepção e contexto de um sistema colaborativo (envio de mensagem, recebimento, notificação, status, etc), assim como é um subsistema de vários outros sistemas colaborativos (como redes sociais, texto colaborativo, entre outros). Desta forma, os microserviços desenvolvidos poderão ser reutilizados por outras aplicações que necessitam de recursos do chat. Além das vantagens de reutilização, existe a possibilidade de um sistema utilizar somente alguns microserviços específicos, fazendo com que não seja obrigatório o uso de todos eles dentro da aplicação proposta. Com relação a percepção, haverá, por exemplo,

microserviços de notificação e leitura de novas mensagens, focado nos aspectos mensagem lida. Já com relação ao contexto, haverá microserviços que fazem o gerenciamento e criação dos grupos de chat, que terão foco nos aspectos de contexto, podendo ser este o grupo de contatos do usuário.

Logo, o objetivo deste trabalho é desenvolver uma arquitetura de microserviços voltada para a percepção da colaboração em sistemas de chat. Aonde os microserviços da aplicação devem ser desenvolvidos como componentes para que possam ser utilizados por outras aplicações e a interface gráfica desenvolvida com características de percepção e contexto como diferencial.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta a fundamentação teórica dos assuntos que compõem a aplicação apresentada nesse artigo. Assim, a subseção 2.1 explica os conceitos da arquitetura de software baseada em microserviços. Na subseção 2.2 é apresentada a arquitetura RESTfull para APIs. A subseção 2.3 trás o conceito de containers para microserviços utilizando Docker. A subseção 2.4 explica os aspectos da percepção para colaboração. Por fim, a subseção 2.5 trás os trabalhos correlatos e os comparar com a aplicação apresentada nesse artigo.

2.1 ARQUITETURA DE MICROSERVIÇOS

Segundo Lewis e Fowler (2014) frustações com a arquitetura de sistemas monolíticos levaram ao início do padrão de microserviços. Microserviços são pequenas aplicações específicas em determinados segmentos e que são estruturadas ao ponto de poderem ser reutilizadas por vários sistemas diferentes (NEWMAN, 2015). Por exemplo, um microserviço de mensageria deverá ser específico em fazer gestão de mensagens e pode ser adicionado modularmente em vários sistemas, pois não tem regras de negócio específicas de algum sistema.

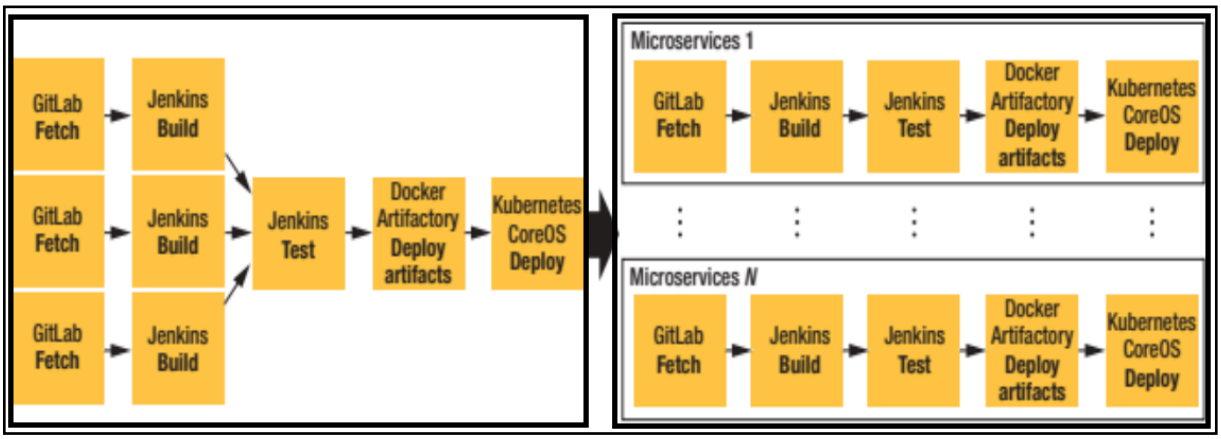
Lewis e Fowler (2014) também afirmam que um ponto que justifica o uso dos microserviços é que eles são implantados e escalam de maneiras diferentes, além de cada microserviço ter sua área de atuação bem definida, o que facilita saber qual o objetivo específico de cada um. O padrão de microserviços não é algo totalmente revolucionário ou novo. Segundo Lewis e Fowler (2014) suas raízes remetem no mínimo aos princípios de design do Unix. O design de arquitetura do Unix é um conjunto de componentes, com dois principais componentes sendo núcleo e programas do sistema, além de uma série de outros componentes desacoplados que trazem funcionalidades para a aplicação.

Desenvolver em uma arquitetura de microserviços em empresas tem muitos benefícios, um deles é que não há a necessidade das equipes reescrevam todo o aplicativo se quiserem adicionar novos recursos (HÁMORI, 2015). Como microserviços são menores, eles também geram menos códigos, tornando a manutenção mais fácil e rápida. Isso economiza esforço e aumenta a produtividade geral (HÁMORI, 2015).

Outro ponto importante no uso de microserviços é a possibilidade de realização de *deploys* automatizados, tendo em vista que após o desenvolvimento de uma aplicação o próximo desafio é subir a aplicação para um ambiente produtivo, sendo que essa metodologia aproxima os times e facilita o *deploy* da aplicação (BALALAIE; HEYDARNOORI; JAMSHIDI, 2014). Com a necessidade de reutilização e a alta demanda sobre *deploy* através de microserviços é necessária a automatização do processo de *deploy* (BALALAIE; HEYDARNOORI; JAMSHIDI, 2014).

Na Figura 1 está ilustrado os fluxos e processos de *deploy* automatizado em uma arquitetura de aplicação monolítica (imagem da esquerda) e de microserviço (imagem da direita). Ainda na Figura 1, os componentes de *deploy* automatizado das duas arquiteturas não diferem muito, o que difere é a forma em que eles são utilizados em cada modelo. O item `GitLab` é o repositório das aplicações, no caso da arquitetura monolítica uma aplicação pode ter vários repositórios, o que indica todas as dependências que são necessárias para a aplicação funcionar. Quando o *deploy* for preparado, todas as dependências da aplicação devem ser incluídas no pacote de *deploy*. Já no caso dos microserviços, cada um tem seu próprio repositório e não tem dependência de outros microserviços para que seja feito o *deploy*. Por isso existe apenas um item `GitLab` no processo de microserviços, enquanto podem ter vários no processo de *deploy* monolítico. Os itens com pré-fixo Jenkins no nome, `Jenkins Build` e `Jenkins Test`, são os responsáveis pelo *pipeline* de *deploy*. Na arquitetura de monolítica, como pode existir mais de um repositório que compõe a aplicação, para cada repositório deve ser feito o processo de *build* da aplicação, o que pode ser demorado e caso quebre algum *build*, quebra o *deploy* inteiro. Já na arquitetura de microserviço, por ser apenas um repositório e um *build* o tempo de *deploy* e a chance de quebra do mesmo é muito menor. O `Jenkins Test` é o item responsável por rodar os testes para liberar o *deploy*. Esses testes podem ser qualquer tipo de teste que seja necessário, por exemplo, pode rodar os testes unitários, teste automatizados e testes estáticos de código antes de executar o processo de *deploy* da aplicação. Para os dois modelos de arquitetura existe apenas um componente `Jenkins Test`. Os itens `Docker Artifactory` e `Kubernetes CoreOS Deploy` funcionam da mesma forma para as duas arquiteturas. Esses itens indicam o processo de colocar a aplicação em container, bem como em ambiente produtivo.

Figura 1 - Comparação arquitetura de deploy monolítico vs microserviços



Fonte: Adaptado de Balalaie, Heydarnoori e Jamshidi (2014).

2.2 ASPECTOS DA PERCEPÇÃO E CONTEXTO PARA A COLABORAÇÃO

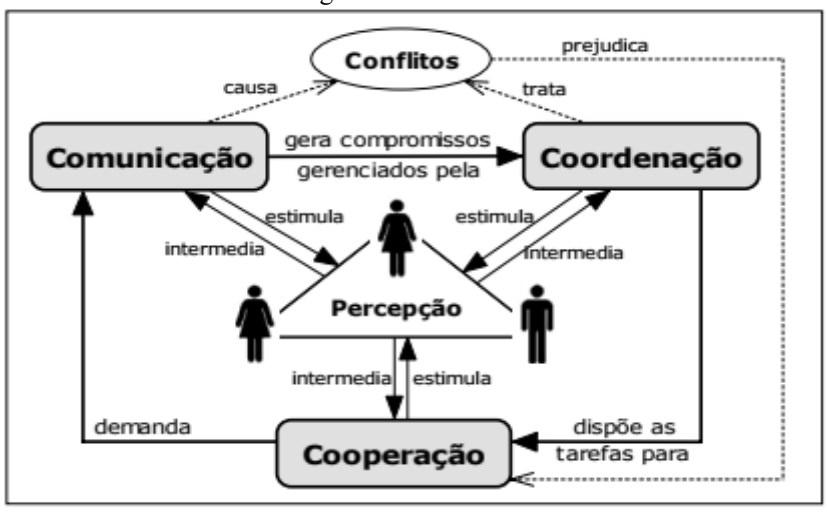
Segundo Pimentel e Fuks (2011), sistemas colaborativos surgiram para integrar equipes, gerir projetos, trocar informações e sempre foram um desafio para as empresas. Esse desafio está relacionado em manter um grupo de pessoas dentro de um mesmo contexto de informações e na forma de notificá-las.

A percepção dentro de sistemas colaborativos está relacionada a capacidade mental de um usuário identificar e compreender ações de um grupo. Já o contexto, ajuda a tornar notificações mais efetivas, pois permite filtrar informações irrelevantes e evita a sobrecarga de informações (PIMENTEL; FUKS, 2011, p.161).

Para que a colaboração seja efetiva, é necessário que seja possível a troca de informações em um espaço compartilhado. As trocas de informações que ocorrem durante a comunicação geram artefatos que devem ser de contexto geral dos usuários, que por sua vez são organizados pela relevância e dispõe as tarefas que são executadas em cooperação (FUKS; RAPOSO; GEROSA, 2003). Quando os indivíduos estão contextualizados, eles têm necessidade de se comunicar para renegociar e tomar decisões sobre situações não previstas inicialmente. Através da percepção, o indivíduo se informa sobre o que está acontecendo, sobre o que as outras pessoas estão fazendo e adquire informações necessárias para seu trabalho (FUKS; RAPOSO; GEROSA, 2003).

A Figura 2 ilustra o Modelo 3C, em que os membros de um grupo que colaboram precisam comunicar-se, cooperar entre si e coordenar as atividades compartilhadas para resolução de um problema. Segundo Pimentel e Fuks (2011), a comunicação no trabalho em grupo é voltada para a ação. Por sua vez, é na comunicação que ocorrem as negociações e tomadas de decisão. Já na coordenação as pessoas lidam com conflitos e organizam atividades para que não tenham desperdícios na comunicação e esforço de cooperação. Renegociar e tomar decisões em situações de imprevistos durante a cooperação demanda comunicação, que demanda coordenação para reorganizar atividades e com as informações da percepção as pessoas recebem *feedback* sobre as atividades das ações dos seus colegas (PIMENTEL; FUKS, 2011). Esses elementos focam nos aspectos mais relevantes da colaboração e os Cs se inter-relacionam para que a colaboração e percepção possam ocorrer (PIMENTEL; FUKS, 2011, p.25).

Figura 2 - Modelo 3C



Fonte: Fuks, Raposo e Gerosa (2003).

Pimentel e Fuks (2011) dizem que a percepção é um fator determinante para a colaboração efetiva e reduz isolamento e solidão no grupo, ainda afirmam que a percepção e conhecimento das atividades do um grupo geram um maior engajamento. Para assimilar o que ocorre no grupo, o usuário deve ter informações a sua disposição que geram a percepção. Um *framework* aderente ao modelo de percepção em grupo é o 5W + 1H. Esse *framework* é composto por 6 perguntas, sendo: quem (*who*) que remete a informação de presença e disponibilidade de uma pessoa no grupo; o que (*what*) que indica informação sobre evento de interesse do grupo; onde (*where*) que remete a localização do evento, onde o evento ocorreu; quando (*when*) que indica a informação sobre tempo do evento, em que momento o evento aconteceu; como (*how*) que indica a informação da maneira como aconteceu o evento; e por que (*why*) indicando intensões e motivações do acontecimento do evento (PIMENTEL; FUKS, 2011, p.159). Pimentel e Fuks (2011) finalizam dizendo que esses mecanismos para gerar percepção apoiam o usuário, mas garantem que o usuário tenha ela, pois a percepção é um estado mental e o que se faz é garantir os mecanismos para tentar gerar esse estado, mas não tem como garantir.

2.3 TRABALHOS CORRELATOS

Esta seção descreve três trabalhos correlatos ao trabalho apresentado neste artigo. O Quadro 1 aborda um trabalho que implementa um gerenciador de conteúdo digital em uma arquitetura de microserviços (VAN GARDEREN, 2010). O Quadro 2 fala sobre uma aplicação para edição de documentos colaborativa desenvolvida na arquitetura de microserviços (GADEA; TRIFAN; IONESCU, 2016). O Quadro 3 detalha a aplicação Slack que é uma ferramenta e chat de equipes que suporta canais, conversas privadas e integração com serviços externos.

Quadro 1 - Arquivematica: Usando microserviço com software open-source para administrar conteúdo digital

Referência	Van Garderen (2010)
Objetivos	Para administrar o conteúdo digital, a plataforma oferece uma alternativa aos sistemas que administram conteúdo em repositórios online. A administração desses conteúdos, segundo Van Garderen (2010), pode ser um problema complexo para pequenas e médias empresas (VAN GARDEREN, 2010). O projeto Archivematica implementou microserviços integrados com ferramentas open source que permitem aos usuários processar conteúdos digitais, bem como acessar e aplicar políticas específicas de preservação de conteúdo.
Principais funcionalidades	Administrar conteúdo digital em repositórios online.
Ferramentas de desenvolvimento	Informação não foi localizada
Resultados e conclusões	O objetivo do projeto era reduzir o custo. Este objetivo foi atingido, pois as estruturas são compartilhadas entre várias aplicações. Outro objetivo atingido foi baixar a complexidade técnica da implantação de uma solução de gerenciamento de conteúdo digital (VAN GARDEREN, 2010).

Fonte: elaborado pelo autor.

Quadro 2 - Arquitetura de microserviços para edição de documentos colaborativos aprimorada com reconhecimento facial

Referência	Gadea, Trifan e Ionescu (2016)
Objetivos	Uma plataforma para editar documentos de forma colaborativa
Principais funcionalidades	As funcionalidades voltadas a alteração de documentos são sincronizadas em tempo real a todos os usuários dentro de uma mesma seção. Dentro dessa mesma seção a aplicação disponibiliza um chat para que as pessoas possam manter a comunicação dentro do próprio aplicativo. A funcionalidade de reconhecimento facial está incorporada dentro do editor de texto e o microserviço responsável pelo reconhecimento facial é chamado automaticamente ao detectar uma imagem dentro do documento de texto. O sistema consiste em uma interface web focada na edição de textos, mas que tem funcionalidade de chat e reconhecimento facial (GADEA; TRIFAN; IONESCU, 2016).
Ferramentas de desenvolvimento	Informação não foi localizada
Resultados e conclusões	Informação não foi localizada

Fonte: elaborado pelo autor.

Quadro 3 - Slack

Referência	Slack Technologies (2014)
Objetivos	Ser um *hub* de informações, centralizando informações utilizadas em tarefas diárias
Principais funcionalidades	O Slack é uma ferramenta e chat de equipes que suporta canais, conversas privadas e integração com serviços externos. Por exemplo, os arquivos do Google Drive e do Dropbox, áudios do SoundCloud e vídeos do YouTube, Hangouts, para a realização de conferências e BitBucket que também colabora com equipes de desenvolvimento de software que utilizam a ferramenta para gestão de código-fonte. Sobre as formas de notificação, o Slack utiliza e-mails, caixas de notificação na aplicação desktop e push de notificação em dispositivos móveis.
Ferramentas de desenvolvimento	Informação não foi localizada
Resultados e conclusões	Informação não foi localizada

Fonte: elaborado pelo autor.

3 WEB CHAT ORIENTADO A MICROSERVIÇOS

Essa seção apresenta o desenvolvimento da web chat baseado em microserviços, sendo este o foco desse artigo. Os Requisitos Funcionais (RF) da aplicação estão apresentados no Quadro 4 e os Requisitos Não Funcionais (RNF) estão apresentados no Quadro 5.

Quadro 4 - Requisitos Funcionais da Aplicação

Requisitos Funcionais
RF01: permitir a realização de cadastro de novos usuários
RF02: permitir a realização de confirmação de cadastro mediante a um código de verificação
RF03: permitir a realização de login mediante um CPF e senha
RF04: permitir que se criem chats em grupos
RF05: permitir que se criem chats privados
RF06: permitir que o usuário visualize sua lista de chats
RF07: permitir que usuário adicione contatos
RF08: permitir que o usuário envie mensagens no chat privado e grupo
RF09: permitir que o usuário veja a data e hora da visualização da mensagem

Fonte: elaborado pelo autor.

Quadro 5 - Requisitos Não Funcionais da Aplicação

Requisitos Não Funcionais
RNF01: ser desenvolvido na linguagem Python (*backend*)
RNF02: ser desenvolvido na arquitetura de microserviços (*backend*)
RFN03: ser desenvolvido no framework Angular (*frontend*)
RNF04: ter interface responsiva (*frontend*)
RNF05: utilizar o banco de dados MySQL
RNF06: ser desenvolvido para ambiente WEB

Fonte: elaborado pelo autor.

Para o desenvolvimento do *backend* da aplicação foi utilizada a linguagem Python na sua versão 3.6 junto ao ambiente de desenvolvimento PyCharm Edu. Já para o *frontend*, foi utilizado o *framework* JavaScript Angular na versão 7.2, utilizando a ferramenta de desenvolvimento Visual Studio Code. Para persistência de dados foi utilizado o banco de dados relacional MySql na versão 8.0. Toda a comunicação entre os microserviços Python e a aplicação Angular de *frontend* foi feita através de requisições HTTP. A comunicação dos serviços com o banco de dados MySql foi feita com auxilio do driver de comunicação do MySql. Todos os serviços e banco de dados rodaram sobre um ambiente Docker configurado.

No desenvolvimento dessa aplicação foi utilizada a tecnologia Docker para desenvolver o ambiente da aplicação. Os containers Docker estão mudando a forma de desenvolver, distribuir e executar softwares. Containers são um encapsulamento para um aplicativo e suas dependências (MOUAT, 2016, p.19). Muito comparados com *Virtual Machines* (VMs), os containers assim como as VMs contêm uma instancia de sistema operacional isolada que é utilizada para executar aplicações, mas os containers possuem usos e vantagens que seriam muito difíceis ou quase impossíveis de executar em VMs tradicionais segundo Mouat (2016).

Segundo Mouat (2016), as principais vantagens na utilização de containers em relação à VMs são:

a) o Docker possui um sistema operacional para gestão dos containers, que os tornam muito mais eficientes, que pode fazer um container ser iniciado ou interrompido em segundos;
b) mover o ambiente de execução de uma aplicação em um container Docker é muito mais fácil e rápido, pois basta que o Docker esteja instalado no ambiente para que o padrão do container seja mantido;
c) a padronização do ambiente com containers Docker permite que os desenvolvedores executem vários containers no seu local de desenvolvimento ao mesmo tempo, assim, emulando um ambiente completo localmente;
d) containers apresentam vantagens para implantação em *cloud*, resolvendo horas de configurações e instalações no ambiente de execução, diminuindo as preocupações com diferenças de ambiente e disponibilidade das dependências.

Mouat (2016) explica os containers Docker em duas principais divisões, Docker Engine e Docker Hub. O Docker Engine oferece a interface para a execução de containers e o Docker Hub como o repositório de imagens de containers pública que permite a reutilização de imagens já construídas por outras empresas e pessoas.

No Quadro 6, tem o arquivo Dockerfile com as configurações do serviço de usuário. Na primeira linha tem-se a importação da imagem Python 3.6.4 em um sistema Alpine Linux 3.7. Essa imagem pode ser encontrada no Docker Hub (Repositório de imagens do Docker). Na segunda linha define-se o diretório principal do container Docker, que para o microserviço de usuário é definido como `ms-user`. Na próxima linha, usando o comando `ADD` é copiado todo o conteúdo que está no diretório `app` da aplicação para o diretório `ms-user/app` do container Docker. O comando `ADD` é um comando similar ao comando `COPY`. Na quarta e quinta linha são executados comandos semelhantes a terceira linha, sendo que na quarta linha é copiado o arquivo `run.py` para o diretório `ms-user` e na quinta linha é copiado o arquivo `requirements.txt` para a pasta `etc`. Na sexta linha é executado o comando Python para instalar as dependências da aplicação no container Docker. Em seguida, o comando define em que porta o container Docker estará disponível e por fim, o comando Python para executar a aplicação dentro do container Docker. O Quadro 6 contém apenas o Dockerfile de uma microserviço da aplicação, mas cada microserviço, webservice, banco de dados e *frontend* dessa aplicação tem o seu próprio Dockerfile configurado.

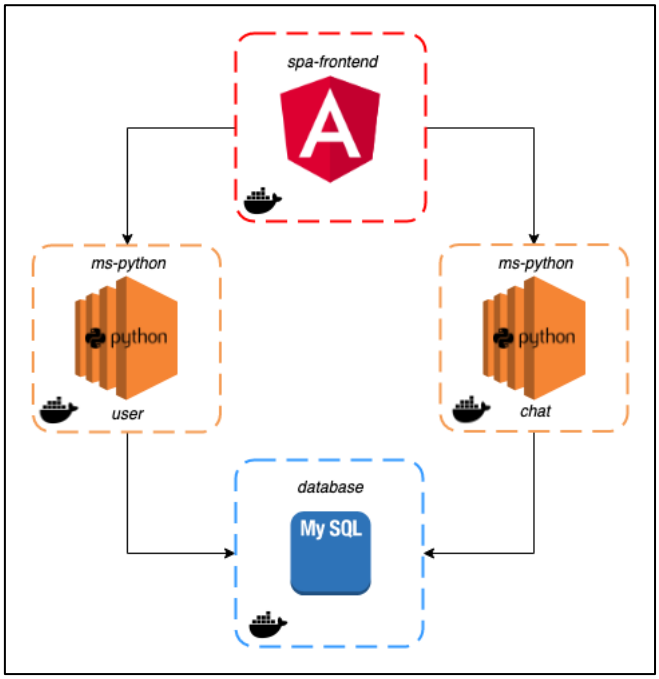
Quadro 6 - Dockerfile microserviço *user*

```
1  FROM python:3.6.4-alpine3.7
2  WORKDIR /ms-user
3  ADD ./app /ms-user/app
4  ADD ./run.py /ms-user
5  COPY ./requirements.txt /etc
6  RUN pip install -r /etc/requirements.txt
7  EXPOSE 8080
8  CMD ["python", "run.py"]
```

Fonte: elaborado pelo autor.

A Figura 3 ilustra a disposição dos principais elementos da aplicação. O item `spa-frontend` é a aplicação em que estão todas as regras de `cliente-server-side`, todos os arquivos HTML e CSS, além de ser a aplicação que o usuário final acessa. Logo em seguida, tem-se os dois microserviços Python, sendo o `user` e `chat`, sendo que neles está toda a regra de negocio do web chat. No ultimo elemento da figura, está o banco de dados que é acessado pelos dois microservicos da aplicação. Essa arquitetura faz com que apenas os serviços que tem relação tenham comunicação, assim segregando as funcionalidades de cada um deles.

Figura 3 - Arquitetura da aplicação

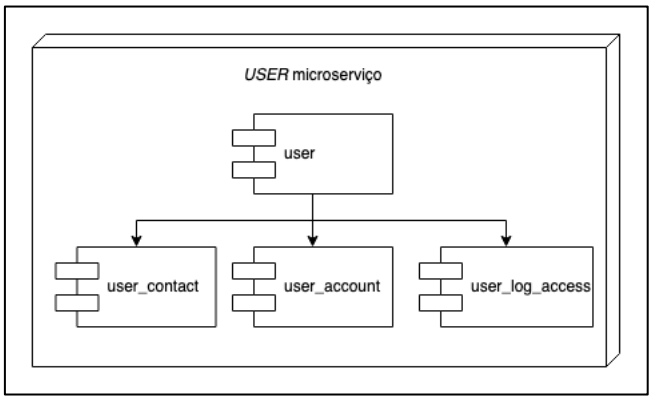


Fonte: elaborado pelo autor.

A Figura 4 ilustra os principais componentes do microserviço que fazem gestão das informações do usuário. Nesse serviço tem-se o principal componente que compõe os dados e validações do usuário, que está indicado no diagrama como `user`.

Relacionado ao componente `user` tem-se o componente `user_contact`. Esse componente faz o vínculo de usuários com seus contatos formando assim sua rede de contatos para iniciar chats privados e grupos. Ele também efetua a validação de usuários cadastrados no sistema ao efetuar a adição de contatos. A criação de contatos é um ponto extremamente importante no fluxo da aplicação, pois é no microserviço chat (que será explicado ainda nesse artigo) que ocorre a criação de uma conversa ou grupo e isso só é possível se o usuário possuir contatos cadastrados. No componente `user_account`, acontece a relação do usuário com a geração de uma conta e a validação da mesma, além da geração de sessão e confirmação de conta após o cadastro. O último componente desse microserviço é o `user_log_access` que é responsável em fazer os controles de acesso de usuário.

Figura 4 - Diagrama de componentes microserviço *user*



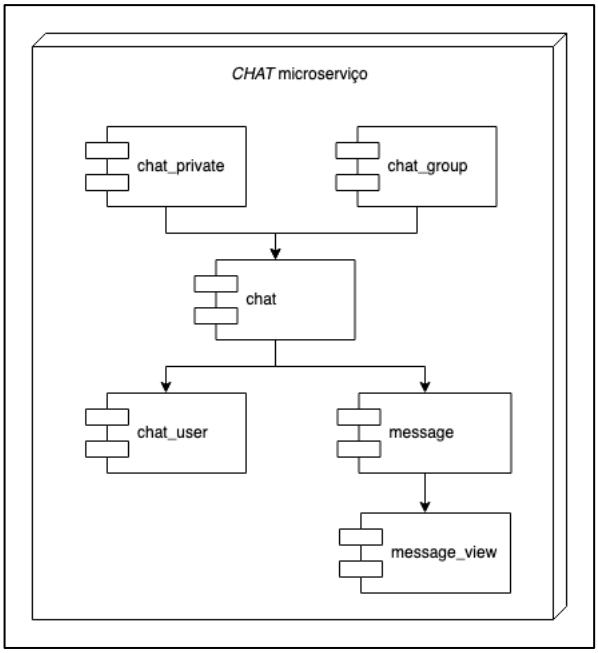
Fonte: elaborado pelo autor.

Na Figura 5, está o diagrama de componentes do microserviço `chat`. Esse serviço é responsável por toda a gestão e criação dos chats da aplicação, criação de grupos e chats privados, além da gestão das mensagens da aplicação.

O principal componente do microserviço de chat é o componente `chat` demonstrado na Figura 6. Ele é responsável por orquestrar todas as operações do serviço. Existem componentes específicos para cada tipo de chat e eles derivam do componente principal. O componente `chat_private` cria e valida os chats de uma pessoa para outra pessoa. Já o componente `chat_group` valida a criação dos grupos de chat. O componente `chat_user` vincula usuários criados no microserviço *user* ao chat, assim, garantindo que cada usuário possa ver somente os chats que ele está vinculado. No

componente `message` é gerida todas as mensagens, envio, recebimento e validação. Em sua composição, o componente `message_view` garante a *feature* de visualização de mensagem, informação relevante para que o usuário possa ver hora e data que outros usuários visualizaram a sua mensagem.

Figura 5 - Diagrama de componentes microserviço chat



Fonte: elaborado pelo autor.

Segundo Fowler (2017), um microserviço deve estar preparado para qualquer catástrofe e deve ser tolerante a falhas. Fowler (2017) afirma que na construção de sistemas distribuídos de grande escala pode ocorrer dos componentes falharem, sendo que eles poderão falhar com frequência. Qualquer cenário de falha pode acontecer em algum momento de vida útil do microserviço e dependendo da complexidade do ecossistema e a forma que a aplicação foi desenvolvida, todo o sistema pode ser comprometido (FOWLER, 2017). A única maneira de mitigar falhas catastróficas e evitar comprometer a disponibilidade de todo o sistema é desenvolver de uma forma que o sistema possa ser tolerante a falhas internas e externas (FOWLER, 2017). Falhas internas podem ser todos os problemas que ocorrem dentro do próprio microserviço e que podem ser contornadas em situações de erro, como por exemplo, um campo que foi mapeado como inteiro e retorna um decimal, é um erro considerado interno. Falhas externas são consideradas falhas em chamadas para algum serviço dependente ou biblioteca externa. Na aplicação descrita nesse artigo foi desenvolvida tolerância a falhas externas, para cobrir possíveis problemas em algum microserviço específico e tentar manter o funcionamento da aplicação mesmo quando algum serviço não estiver disponível.

No Quadro 7 está o código da aplicação descrita nesse artigo que trata de tolerância a falha de uma situação de dependência externa. Esse código é o microserviço chat, que faz uma HTTP Request para o serviço de usuário (`linha 1`). O microserviço de chat recupera os dados do usuário para montar a interface com dados do chat mais informações do usuário. A tolerância a falha dessa situação foi feita para quando o microserviço de usuário não estiver disponível (`linhas 10, 11 e 12`), o chat continue funcionando. Caso o serviço de usuário não esteja disponível e o usuário já tenha efetuado login, ele poderá continuar enviando mensagens normalmente e só o que será impactado pela falha do microserviço do usuário de modo que os dados do usuário não aparecerão no chat. Ainda no Quadro 7, a variável `request_user` (`linha 1`) é o conteúdo da chamada para o microserviço de usuário. Caso o parâmetro `status_code` de resposta for 200 (Sucesso) (`linha 2`) ou 404 (Registro não encontrado) (`linha 5`) o sistema funcionará de forma normal, pois são as respostas esperadas pelo serviço. Qualquer divergência desses dois casos, o microserviço de usuário deve estar passando por alguma instabilidade, então, trata-se os dados do usuário como vazios e segue o fluxo da aplicação sem maiores problemas, pois esse dado não impacta no envio e recuperação de mensagens.

Quadro 7 - Código que apresenta desenvolvimento de tolerância a falha

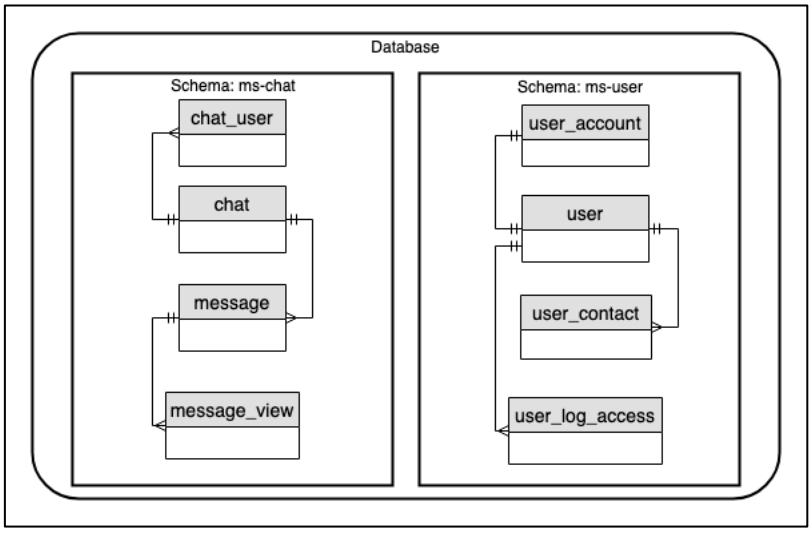
```
request_user = requests.get(self.host + ':8080/ms-user/' + user_cpf)
        if request_user.status_code == 200:
            response_user = json.loads(str(request_user.content, 'utf-8',
errors='replace'))
            self.set_name_private(response_user["fullname"])
        elif request_user.status_code == 404:
            errors.append({"field":"user_cpf", "code":301, "message":"User " +
user_cpf + " not registered"})
        else:
            self.set_name_private("")
            errors.append({"field":"user_cpf", "code":301, "message":"Can't connect
in ms-user to find cpf" + user_cpf})
```

Fonte: elaborado pelo autor.

A arquitetura de banco de dados em uma aplicação orientada a microserviços é um ponto de muita atenção. O ideal é que cada microserviço tenha sua própria estrutura de banco de dados. Na aplicação descrita nesse artigo tem-se um banco de dados compartilhado para os dois microserviços desenvolvidos. Contudo, há uma separação por *schemas* para cada serviço.

Na Figura 6 está o diagrama de entidade e relacionamento dos dois microserviços. No *schema* `ms-chat` estão os dados dos chats e mensagens do usuário. Na tabela `chat` encontram-se todos os chats registrados na aplicação. Junto a tabela `chat_user` é possível localizar quais usuários estão vinculados ao chat. Na tabela `message` encontram-se todas as mensagens vinculadas a um chat e a um usuário. Por último, a tabela `message_view` garante a visualização da mensagem enviada pelos outros usuários que estão no chat. No *schema* `ms-user` está toda a estrutura de dados que suporta o microserviço *user*. Todos os dados de cadastro do usuário estão na tabela `user`. Na tabela `user_account` estão os dados referentes a *login* do usuário. Já na tabela `user_log_access` estão todos os dados referentes a *log* de *login* efetuados pelo usuário, além da informação relacionada a quando foram efetuadas entradas do usuário no sistema. A última tabela do *schema* é a `user_contact`, sendo que nessa tabela ficam registrados os contatos registrados pelo usuário.

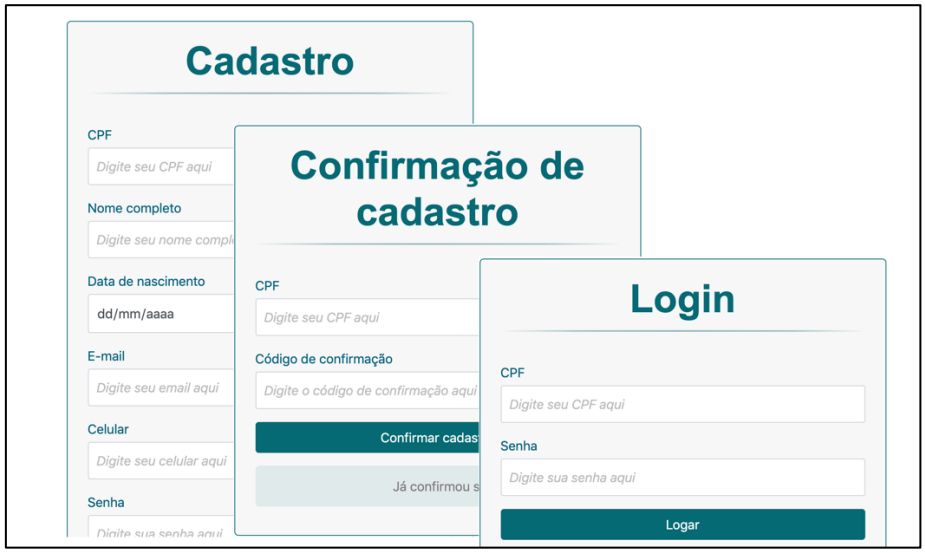
Figura 6 - Modelo de entidade e relacionamento



Fonte: elaborado pelo autor.

Na Figura 7, tem-se as telas não autenticadas que o usuário pode acessar. Usuários já cadastrados podem acessar diretamente o login e o chat. Já os usuários não cadastrados vão seguir para a tela de cadastro. Na tela de cadastro, o usuário preenche suas informações e passa para a tela de confirmação de conta. O usuário irá receber um e-mail com um código de confirmação. Confirmando a conta, ele poderá acessar a aplicação por meio de login.

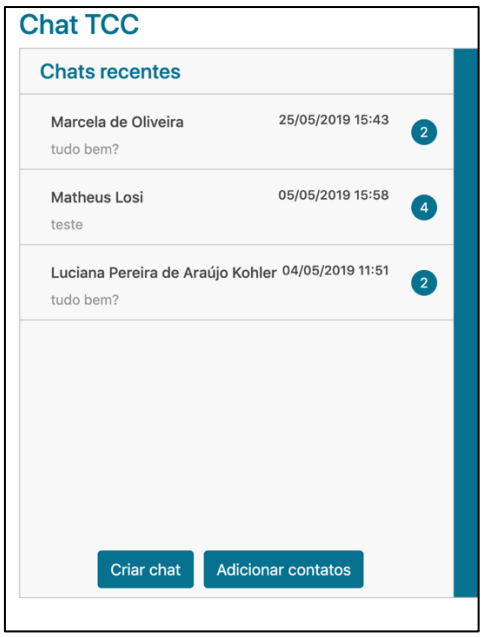
Figura 7 - Telas login, confirmação de cadastro e login



Fonte: elaborado pelo autor.

Ao acessar a aplicação, o usuário pode ter acesso ao seu histórico de chats, criar novos chats, adicionar contatos e mandar mensagens. Na Figura 8, tem-se a listagem de chats do usuário. Após efetuar *login*, o usuário tem acesso a todos os chat e mensagens já criados por ele. Ainda na Figura 10, está uma implementação de percepção, item que notifica o usuário de novas mensagens. O funcionamento é bem simples, mostrar a quantidade de mensagens não lida em cada um dos chats. Contudo, isso trás o ponto de percepção do usuário para aumentar a relevância do chat que deve ser clicado, ou seja, o chat que tem novas informações ainda não vistas pelo usuário.

Figura 8 - Listagem de chats

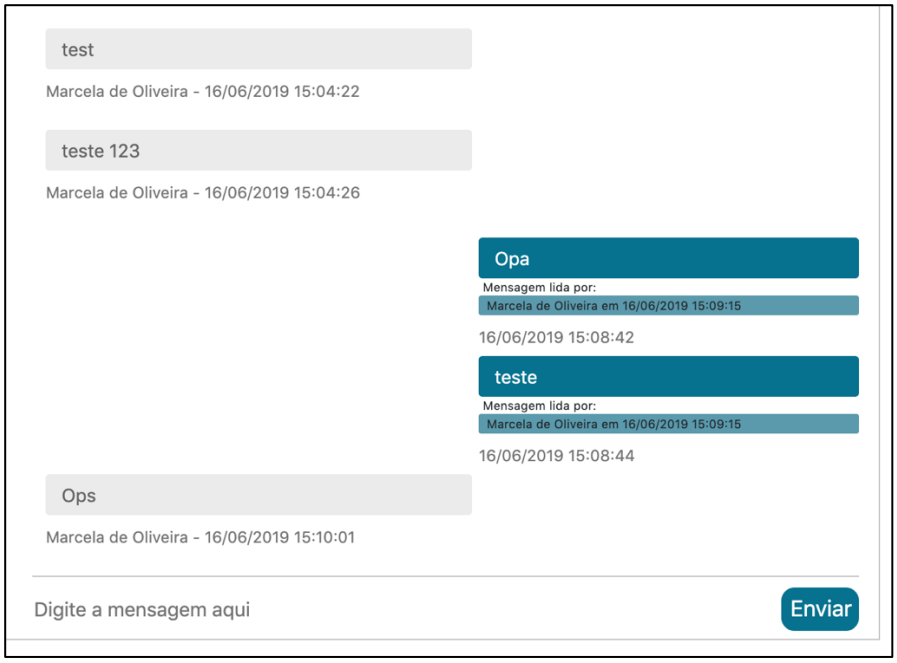


Fonte: elaborado pelo autor.

Também na Figura 8, tem-se as funcionalidades de criação de chat e inserção de novos contatos. Na criação de chats um *modal* abre com a possibilidade de criar um chat privado (somente com uma pessoa) ou grupo (com mais de 2 pessoas). Para conseguir criar algum dos tipos de chat há a necessidade de contatos adicionados.

Na Figura 9, tem-se mais uma funcionalidade de percepção. Essa funcionalidade mostra quem visualizou e que horas visualizou a mensagem. Ainda na Figura 8 existem itens que podem ser correlacionados com o *framework* 5W+1H, sendo que a mensagem lida se encaixa no item do *framewok* "quem (*How*)", apresentando quem leu a mensagem e as cores das mensagens auxiliam no mesmo item, quem enviou. O item "quando (*When*)", pode ser percebido nessa imagem, quando foi lida a mensagem e também quando foi enviada pelo próprio usuário, dados de contexto que auxiliam na percepção e melhoram a comunicação no uso do chat.

Figura 9 - Mensagem visualizada



Fonte: elaborado pelo autor.

4 RESULTADOS

Em relação aos trabalhos correlatos da aplicação apresentada nesse artigo, no Quadro 8 tem-se a comparação das características dessas aplicações com a apresentada neste artigo. No quadro é demonstrado que a ferramenta Slack Technologies (2014) é a aplicação mais próxima da descrita nesse artigo, tirando os itens de disponibilização dos serviços publicamente e modularização pala utilização em outros sistemas. Já as aplicações de Van Garderen (2010) e Gadea, Trifan e Ionescu (2016) trazem como seu principal ponto de correlação a arquitetura de microserviços.

Quadro 8 - Comparação de correlatos

Correlatos / Características	Van Garderen (2010)	Gadea, Trifan e Ionescu (2016)	Slack Technologies (2014)	Losi (2019)
Utilização de arquitetura de microserviços	Sim	Sim	Não	Sim
Possui ferramenta de chat	Não	Sim	Sim	Sim
Ferramenta pública	Não	Não	Sim (Com limitações de uso)	Sim
Ferramenta colaborativa	Sim	Sim	Sim	Sim
WEB aplicação	Não	Não	Sim	Sim
Possui design responsivo	Não	Não	Sim	Sim (Com limitações de dispositivos)
Disponibiliza publicamente seus serviços para integrações com outros sistemas	Não	Sim	Não	Sim

Fonte: elaborado pelo autor.

Um dos pontos descrito no Quadro 8 e um dos principais tópicos desses trabalho é a arquitetura de microserviços. Apenas a aplicação Slack Technologies (2014) não utiliza a arquitetura de microserviços, mas mesmo assim, como os outros pontos de correlação são muito fortes entre a aplicação desse artigo e a aplicação Slack Technologies (2014), ela ainda é a principal ferramenta de comparação. Outro ponto importante descrito no Quadro 8 é se as aplicações possuem ferramenta de chat. Esse ponto é o aspecto principal do assunto de comunicação e percepção da aplicação desse artigo e apenas a aplicação de Van Garderen (2010) não possui essa característica.

O aspecto de disponibilidade pública dos seus serviços para integração com outras aplicações é algo possível para aplicação descrita nesse artigo por meio de integração as APIs Rest da aplicação que são públicas, assim possibilitando o uso por outras aplicações. Já os correlatos apresentados são aplicações sem APIs públicas para reutilização dos seus serviços por outras aplicações.

Para testar a aplicação, foi efetuado um teste com um grupo de alunos utilizando a aplicação descrita nesse artigo. O teste foi realizado na disciplina de Programação III com os alunos do curso de Ciência da Computação. Essa disciplina estuda a programação para web envolvendo tanto *frontend* quanto *backend*. Dessa forma, o público que testou a aplicação tinha os conhecimentos suficientes para valida-la.

Para o teste foi liberada a aplicação na rede interna da FURB e todos puderam acessar dos seus dispositivos, sendo notebooks, computadores de mesa ou dispositivos móveis. Foram aproximadamente 30 minutos de testes e os alunos puderam realizar seu cadastro, adicionar contatos, criar grupos de conversas e enviar mensagens. Ao todo participaram do teste vinte e um alunos. Ao final do teste, foi solicitado que os alunos respondessem a um questionário de usabilidade, contudo somente sete alunos o responderam. Somente sete pessoas responderam, pois, só elas conseguiram utilizar o chat. Isso ocorreu, pois o sistema não havia sido testado com uma grade quantidade de pessoas e não foi disponibilizado em um servidor e sim de uma máquina local acessada pelos alunos que realizaram os testes. Dos sete alunos que realizaram os testes, todos fizeram cadastro e *login* e apenas três criaram chats e grupos, nem todos conseguiram utilizar 100% do chat por sobrecarga da máquina local que estava rodando a aplicação. Outro ponto que colaborou para a sobrecarga nos testes foi a quantidade de chamadas para o *backend* ocasionadas pelo não uso de um *webhook* para a captação de novas mensagens pelo *frontend*, assim, cada usuário conectado fazia diversas chamadas simultâneas para recuperar novas mensagens no chat. Um último ponto que auxiliou na falta de performance da aplicação, foi o fato das mensagens não virem paginadas do *backend*, assim, quanto mais mensagens tivesse o chat, maior o tamanho da requisição.

O questionário aplicado foi sobre a aplicação e as percepções do avaliador em relação ao trabalho. O questionário foi construído com base no *check-list* de usabilidade para web que avalia pontos da interface gráfica conforme algumas normas de usabilidade (SAPO, 2016) e foi disponibilizado em um Google Forms conforme pode-se ver na íntegra no Apêndice B. Foram feitas 14 perguntas ao todo, sendo que 10 foram perguntas com opção de resposta se atendia ou não o quesito, 3 foram perguntas com respostas abertas e a última pergunta foi sobre o contato do usuário com aplicativos de chat como o WhatsApp. As 10 perguntas relacionadas a usabilidade podem ser visualizadas no Quadro 9.

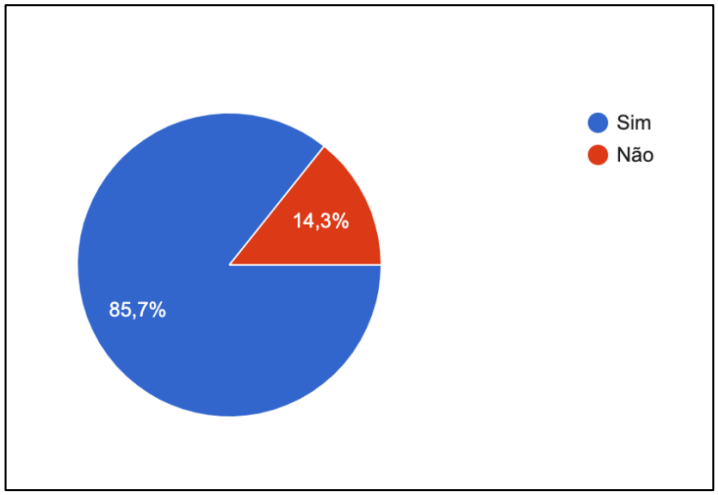
Quadro 9 - Perguntas sobre usabilidade

É sempre fornecido feedback sobre as ações do utilizador?
É fornecido feedback sobre a localização do utilizador?
Os títulos dos links e menus são claros e perceptíveis?
Os itens clicáveis têm aspeto clicável e diferente do resto do conteúdo?
Os itens não clicáveis não se parecem com links ou botões?
O texto dos links faz sentido quando lido fora do contexto?
Os ícones usados são consistentes com as ações que executam?
A informação crítica (que requer a atenção do utilizador) tem destaque suficiente na página?
Existe um contraste suficiente entre a cor dos textos e a cor de fundo?
As mensagens de erro estão junto dos elementos que contêm o erro?

Fonte: elaborado pelo autor.

Na Figura 10, tem-se as respostas da seguinte pergunta “É sempre fornecido feedback sobre as ações do utilizador?”. Essa pergunta foi respondida por 7 pessoas e apenas uma pessoa respondeu que não foi fornecido feedback sobre a ação. Nesse caso foi um *bug* que foi pego na fase de testes, que a requisição para recuperar as novas mensagens falhou e teve que ser atualizada a página para ser atualizada a tela. Logo após o teste, essa situação foi corrigida de acordo com o feedback do usuário.

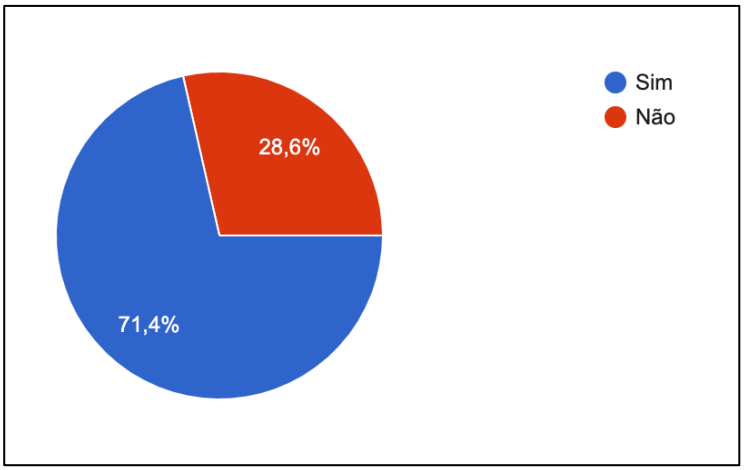
Figura 10 - Respostas da pergunta: É sempre fornecido feedback sobre as ações do utilizador?



Fonte: elaborado pelo autor.

Na Figura 11, tem-se as respostas da pergunta “Os títulos dos links e menus são claros e perceptíveis?”. Essa pergunta também foi respondida por 7 pessoas e 2 pessoas responderam que não foi claro. Em seus comentários o foco foi no posicionamento de alguns botões, que poderiam estar mais claros e que alguns rótulos não estavam claros, como por exemplo na criação do chat tinha a opção de criação do chat `privado` e `grupo` e não era claro para o usuário o que era um chat privado (chat para conversa com apenas uma pessoa). Com base nesses comentários foram realizadas as seguintes alterações: para os botões foi utilizado o padrão da biblioteca Material Design e para os rótulos foram alterados para melhorar o entendimento, no exemplo da criação do chat os rótulos ficaram `criação de chat` e `grupo`.

Figura 11 - Respostas da pergunta: Os títulos dos links e menus são claros e perceptíveis?

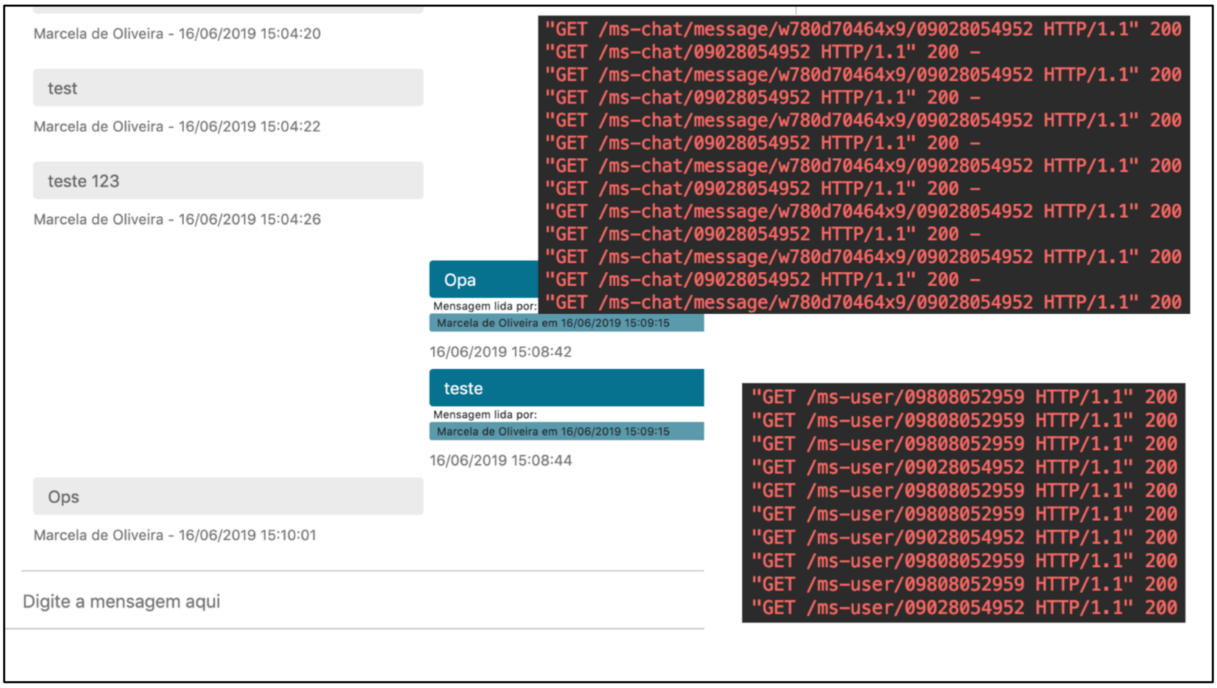


Fonte: elaborado pelo autor.

Outros itens que foram levantados no questionário foram em relação a leiaute. Um ponto sobre a falta de uso de ícones para auxiliar no entendimento das telas e sobre os feedbacks de erros nos campos em formulários foram levantados. O fato de só validar no final da operação alguns campos pode não trazer a melhor experiência. Algumas validações em tempo de preenchimento foram implementadas após os feedbacks da pesquisa.

Um dos itens que não foi testado com o grupo de alunos foi a tolerância a falhas, mas esse teste foi realizado em um ambiente controlado e seus resultados foram satisfatórios. Na Figura 12 são apresentadas as imagens do web chat desse artigo em seu funcionamento normal, com todos os serviços online. Na direita da figura tem-se o status OK (200) das requisições sendo feitas para os dois microserviços da aplicação.

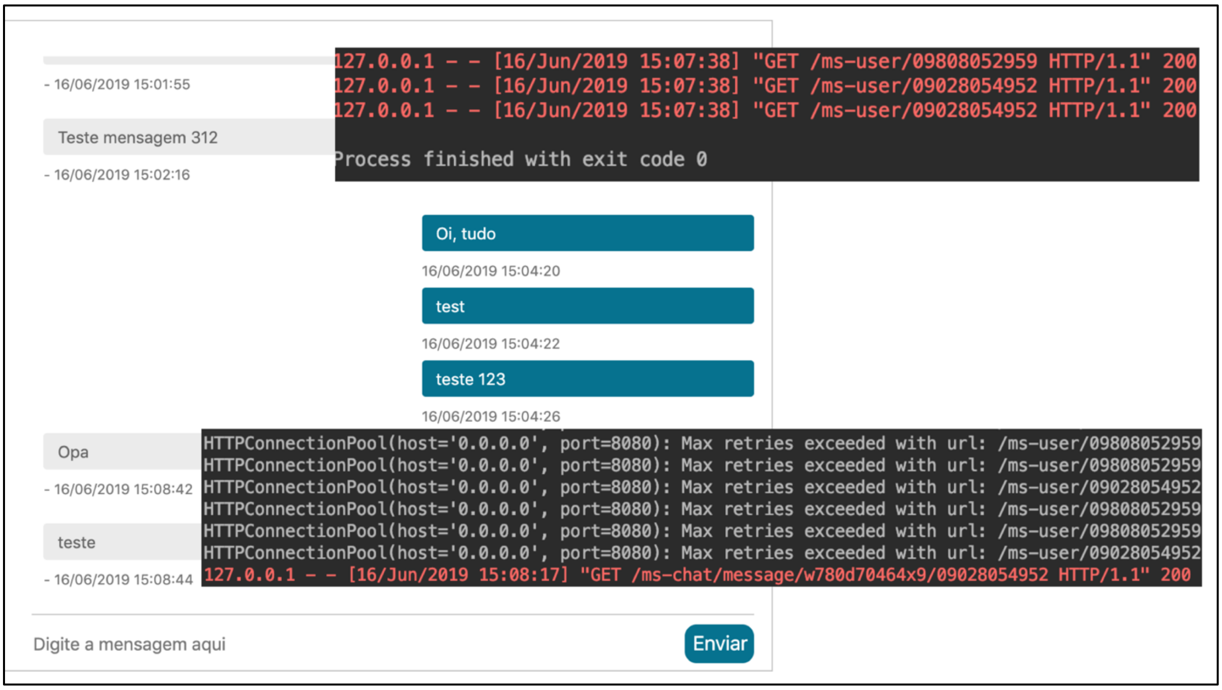
Figura 12 - Aplicação 100% funcional com todos serviços online



Fonte: elaborado pelo autor.

Para testar a tolerância a falhas, foi retirado do ar o microserviço `ms-user`. Na Figura 13 tem-se o resultado após essa configuração. O que indica que esse serviço não está mais em funcionamento é o código de finalização de processo (0) no final do primeiro quadro preto da direita. Mesmo com o microserviço de usuário sem funcionamento, o microserviço `ms-chat` continua funcionando e de acordo com que indica no log do serviço, segundo quadro preto na Figura 13, todas as requisições que o serviço para o microserviço `ms-user`, estão dando erro, mas na última linha desse mesmo quadro, uma requisição continua dando HTTP Status Code 200, o que indica sucesso da operação. Essa requisição é a de envio de mensagem. Ainda na Figura 13, o chat exibido continua funcionando, mas não no seu pleno funcionamento, sendo que algumas informações de usuário que não puderam ser recuperadas aparecem em branco (Linha abaixo da mensagem aonde aparece o nome do usuário), mas o envio de mensagem continua ocorrendo normalmente.

Figura 13 - Aplicação rodando com o serviço ms-user fora do ar



Fonte: elaborado pelo autor.

5 CONCLUSÕES

Esse artigo apresentou o desenvolvimento de um web chat baseado na arquitetura de microserviços e a interface gráfica com aspectos da percepção. É uma aplicação que pode ser publicada na web e ficar aberta para receber cadastros de usuários, sendo que esses usuários cadastrados podem utilizar todas as funcionalidades do chat, criação de conversas, adicionar contatos e enviar mensagens. Para o desenvolvimento do *frontend* da aplicação foi utilizado o ambiente de desenvolvimento Visual Studio Code com a linguagem JavaScript e o *framework* Angular e para o *backend* foi utilizado o ambiente de desenvolvimento PyCharm e a linguagem Python. Tanto *backend* como *frontend* foi utilizada a tecnologia Docker para disponibilizar as aplicações.

Um dos objetivos deste trabalho era desenvolver uma arquitetura de microserviços voltada para a percepção da colaboração em sistemas de chat. Esse objetivo foi alcançado com o web chat descrito nesse artigo, pois foram desenvolvidos dois microserviços de *backend* em Python e uma aplicação *Single Page Application* (SPA) no *frontend* desenvolvidas em Javascript com o *framework* Angular. Outro objetivo desse trabalho era poder ser utilizado por outras aplicações. Para que pudesse ser utilizado por outras aplicações, a aplicação desse artigo foi desenvolvida em forma de uma API Rest, assim cumprindo o objetivo de reutilização. O último objetivo do trabalho descrito nesse artigo era a interface gráfica com características de percepção e contexto como diferencial. Esse objetivo foi cumprido com o desenvolvimento de *features* no *frontend* voltadas para a percepção. Foram desenvolvidos itens como quantidade de mensagens não lidas nas listas de chats, diferenciação de mensagens enviadas e mensagens recebidas, horário de envio e leitura das mensagens e padrão de feedbacks das ações do chat utilizando Material Design, lista de contatos, chats e grupos apenas com contatos adicionados na sua lista, assim os itens de percepção e contexto foram cumpridos.

Em comparação com seus correlatos, a aplicação disponibiliza as principais funcionalidades que são fornecidas por aplicações comerciais de destaque no mercado, oferecendo estrutura modular e expansível para reutilização dos serviços de *backend* dentro de qualquer outra aplicação de forma gratuita.

Como contribuições desse trabalho, ficam os estudos e análises sobre a utilização da arquitetura de microserviços para a componentização e modularização de serviços que podem ser reutilizados por outras aplicações. Além disso, a aplicação dos elementos de contexto e percepção em uma interface gráfica e os microserviços de chat que foram desenvolvidos, sendo que esses poderão ser reutilizados em outras aplicações.

As limitações desse trabalho estão vinculadas a algumas funcionalidades do web chat que poderiam ampliar o uso da ferramenta, download e upload de qualquer tipo de arquivo poderia incrementar o trabalho e dar mais funcionalidade para os usuários. Na parte técnica dois itens poderiam ser evoluídos, sendo que na comunicação entre cliente e servidor para a troca de mensagens poderia ser utilizado um *websocket* para evitar uma quantidade alta de requisições ao servidor. Com a implementação de um *websocket* o cliente sempre seria avisado quando tivesse uma nova mensagem e não precisaria ficar fazendo chamadas sequenciais para verificar se tem novas mensagens a serem carregadas. Outro ponto técnico que poderia ser implementado entre a comunicação dos microserviços é uma aplicação chamada *circuit breaker*. Um *circuit breaker* em uma arquitetura de microserviços faz com que o item de tolerância a falha fique 100% coberto. Ele pode manter em cache de algumas requisições entre microserviços, assim, fazendo que em algum caso de queda de um serviço dependente, caso haja cache, nem se perceba a queda e traga todas as informações necessárias para completar a requisição mesmo com serviço responsável pela informação estando *offline*.

REFERÊNCIAS

BALALAIE, Armin; HEYDARNOORI, Abbas; JAMSHIDI, Pooyan. **Microservices Architecture Enables DevOps.** London: Sharif University of Technology, 2014.

FOWLER, Susan. **Microserviços prontos para produção.** São Paulo: Novatec Editora Ltda, 2017.

FUKS, Hugo; RAPOSO, Alberto Barbosa; GEROSA, Marco Aurélio. **Do Modelo de Colaboração 3C à Engenharia de Groupware.** Salvador: Simpósio Brasileiro de Sistemas Multimídia e Web, 2003.

GADEA, Cristian; TRIFAN, Mircea; IONESCU, Dan. **A Microservices Architecture for Collaborative Document Editing Enhanced with Face Recognition.** 2016. 446 f. Monografia (Especialização) - Curso de Computação, University Of Ottawa, Ottawa, 2016.

HÁMORI, Ferenc. **How Enterprises Benefit From Microservices Architectures.** 2015. Disponível em: https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures/. Acesso em: 29 jun. 2019.

LEWIS, James; FOWLER, Martin. **Microservices: a definition of this new architectural term.** 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 29 jun. 2019.

MOUAT, Adrian. **Usando Docker.** São Paulo: Novatec Editora Ltda, 2016.

NEWMAN, Sam. **Building Microservices.** Sebastopol: O'reilly Media, Inc., 2015.

PIMENTEL, Mariano; FUKS, Hugo. **Sistemas Colaborativos.** São Paulo: Elsevier Editora Ltda, 2011.

SAPO. **Usabilidade (Web).** [2016?]. Porgual. Disponível em: https://ux.sapo.pt/checklists/usabilidade/. Acesso em: 29 jun. 2019.

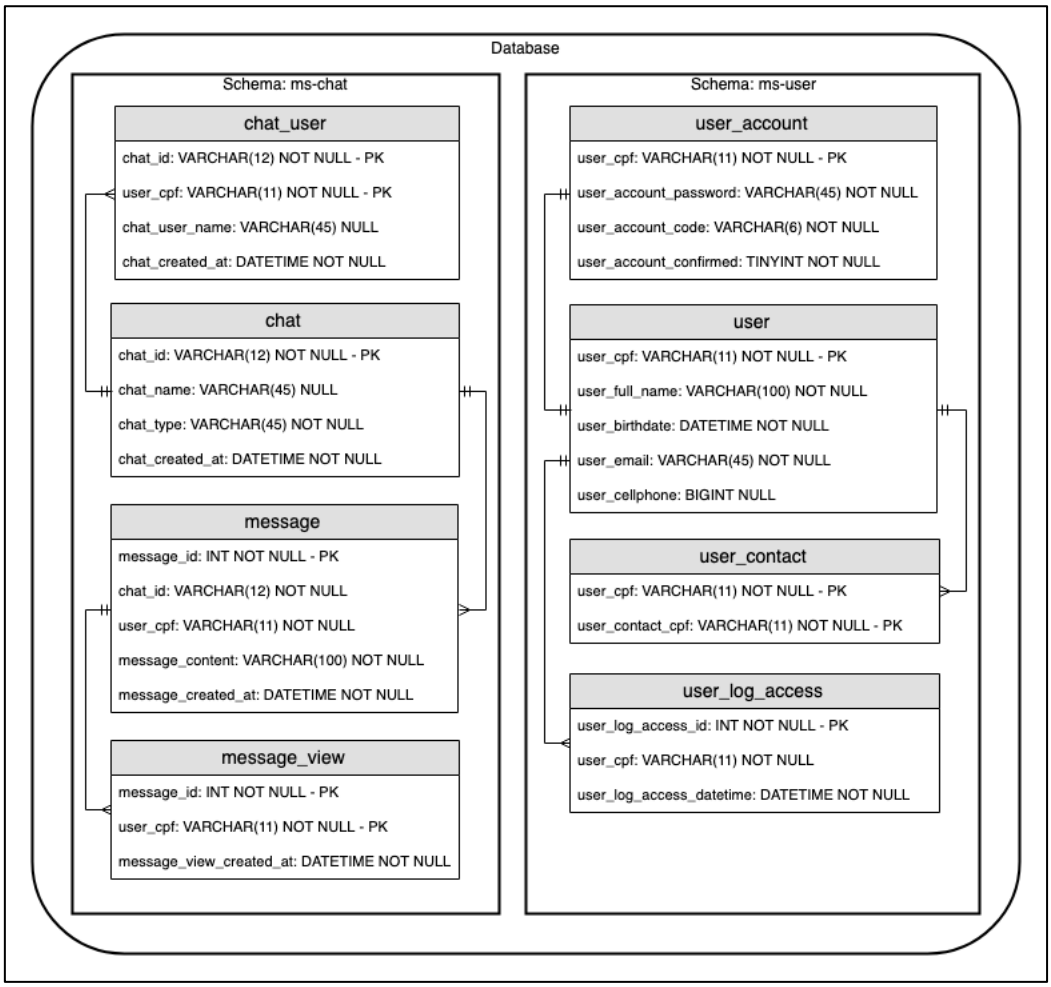
SLACK TECHNOLOGIES (Org.). **Slack.** Vancouver: 2014. Disponível em: https://slack.com/features. Acesso em: 31 ago. 2017.

VAN GARDEREN, Peter. Archivematica: Using Micro-Services And Open-Source Software To Deliver A Comprehensive Digital Curation Solution. **7th International Conference On Preservation Of Digital Objects** (ipres2010). Viena, 5p. 19 set. 2010.

APÊNDICE A – MODELO DE ENTIDADE E RELACIONAMENTO

Na Figura 14 é apresentado o diagrama de entidade e relacionamento completo da aplicação apresentada nesse artigo.

Figura 14 - Diagrama entidade relacionamento completo

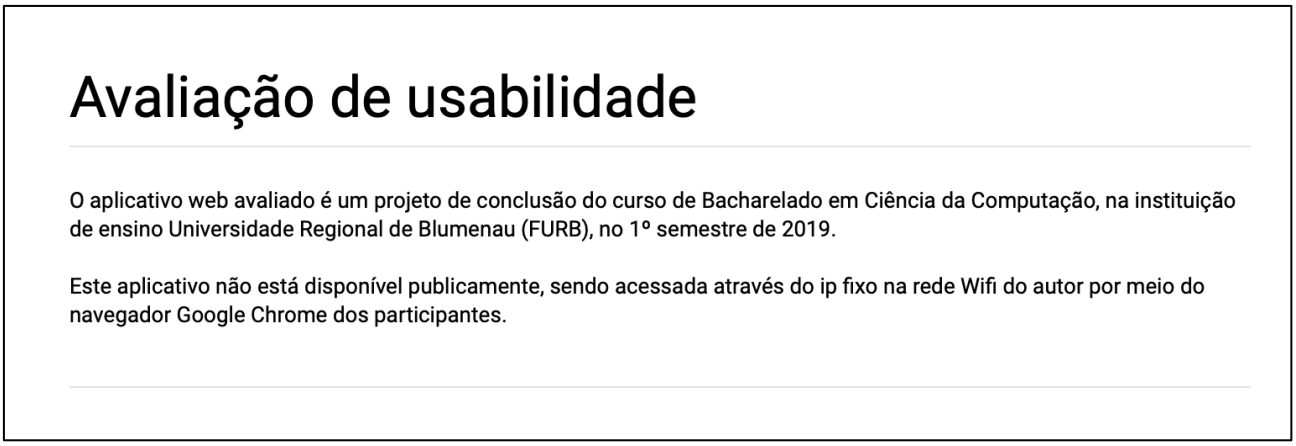


Fonte: elaborado pelo autor.

APÊNDICE B – QUESTIONÁRIO DE USABILIDADE APLICADO COM A TURMA DE PROGRAMAÇÃO WEB

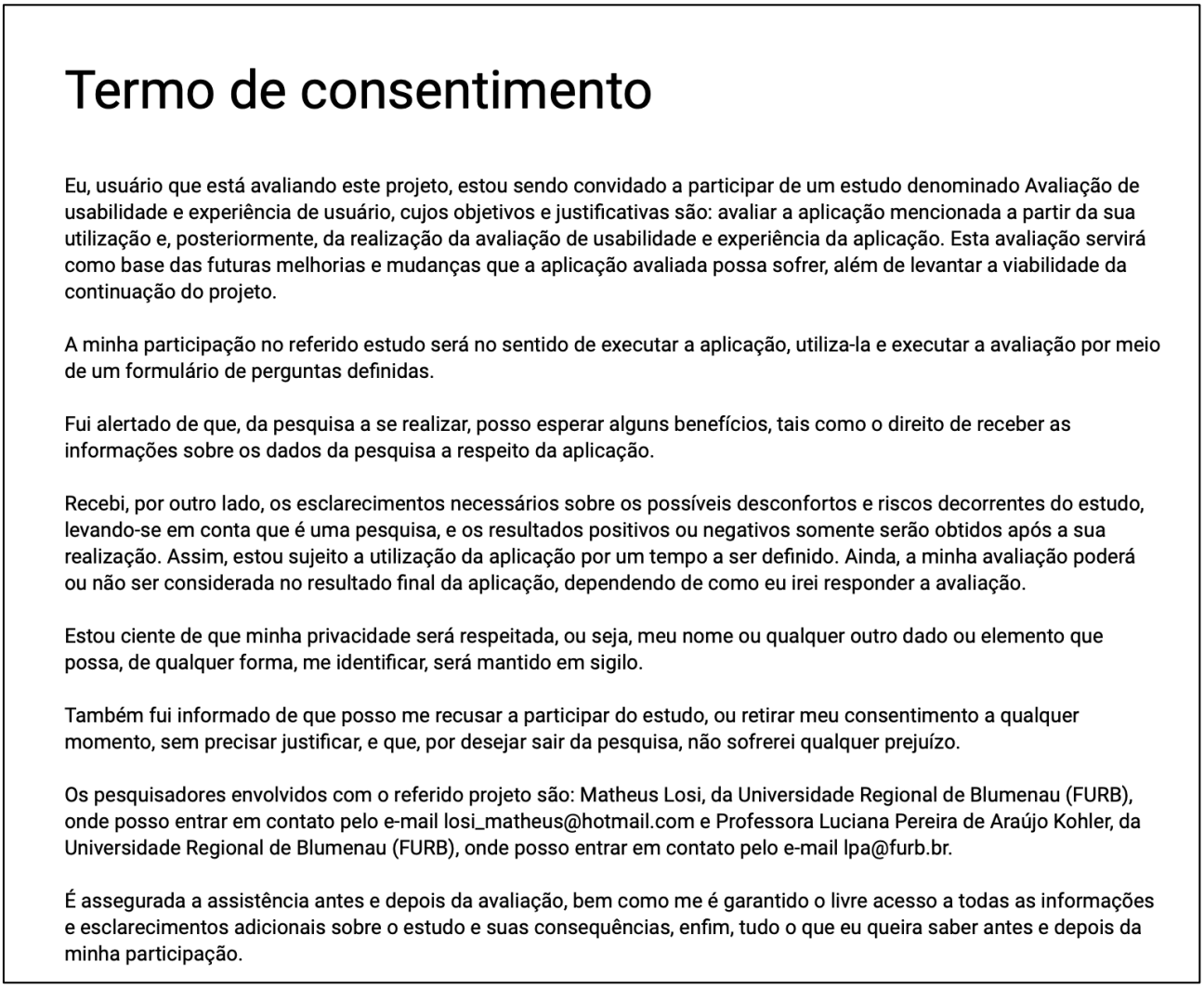
Nesse apêndice está o questionário de usabilidade completo que foi aplicado nos alunos da turma de programação web que testaram a aplicação desse artigo.

Figura 15 - Explicação da avalição de usabilidade



Fonte: elaborado pelo autor.

Figura 16 - Termo de consentimento para realização do questionário



Fonte: elaborado pelo autor.

Figura 17 - Continuação do termo de consentimento

Enfim, tendo sido orientado quanto ao teor de todo o aqui mencionado e compreendido a natureza e o objetivo do já referido estudo, manifesto meu livre consentimento em participar, estando totalmente ciente de que não há nenhum valor econômico, a receber ou a pagar, por minha participação.

Em caso de reclamação ou qualquer tipo de denúncia sobre este estudo devo entrar em contato com Professora Luciana Pereira de Araújo KOhler, da Universidade Regional de Blumenau (FURB), onde posso entrar em contato pelo e-mail lpa@furb.br.

Blumenau, 10 de maio de 2019.

Matheus Losi, Acadêmico - Universidade Regional de Blumenau (FURB)
Luciana Pereira de Araújo Kohler, Professora - Universidade Regional de Blumenau (FURB)

AO PROSSEGUIR PARA A PRÓXIMA SEÇÃO DESTE FORMULÁRIO DE AVALIAÇÃO, DECLARO QUE ESTOU DE ACORDO COM OS TERMOS EXPLÍCITOS ACIMA.

Fonte: elaborado pelo autor.

Figura 18 - Pergunta 1 e 2 do questionário

Avaliação Web - Navegação e Feedback

Esse questionário tem como base o check list de usabilidade para web de https://ux.sapo.pt/checklists/usabilidade/

1) É sempre fornecido feedback sobre as ações do utilizador *

- ○ Sim
- ○ Não

1.1) Comentários

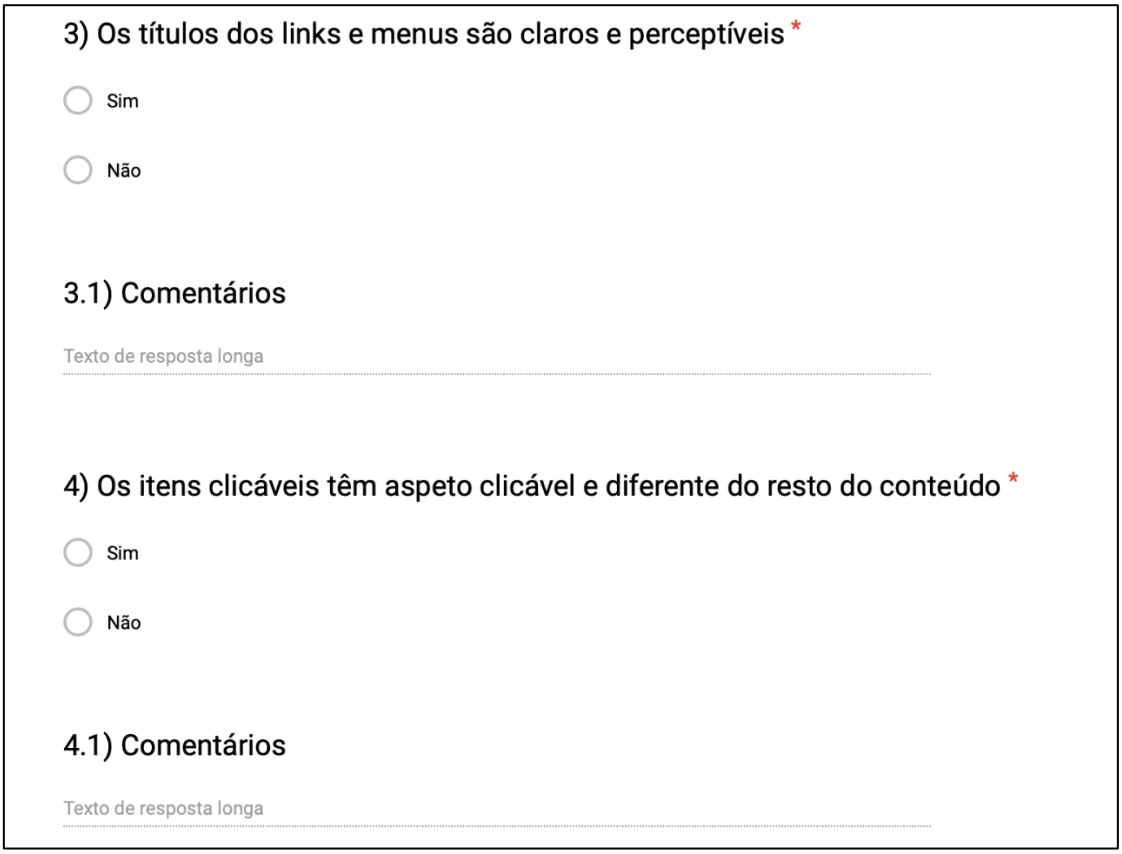
Texto de resposta longa

2) É fornecido feedback sobre a localização do utilizador *

- ○ Sim
- ○ Não

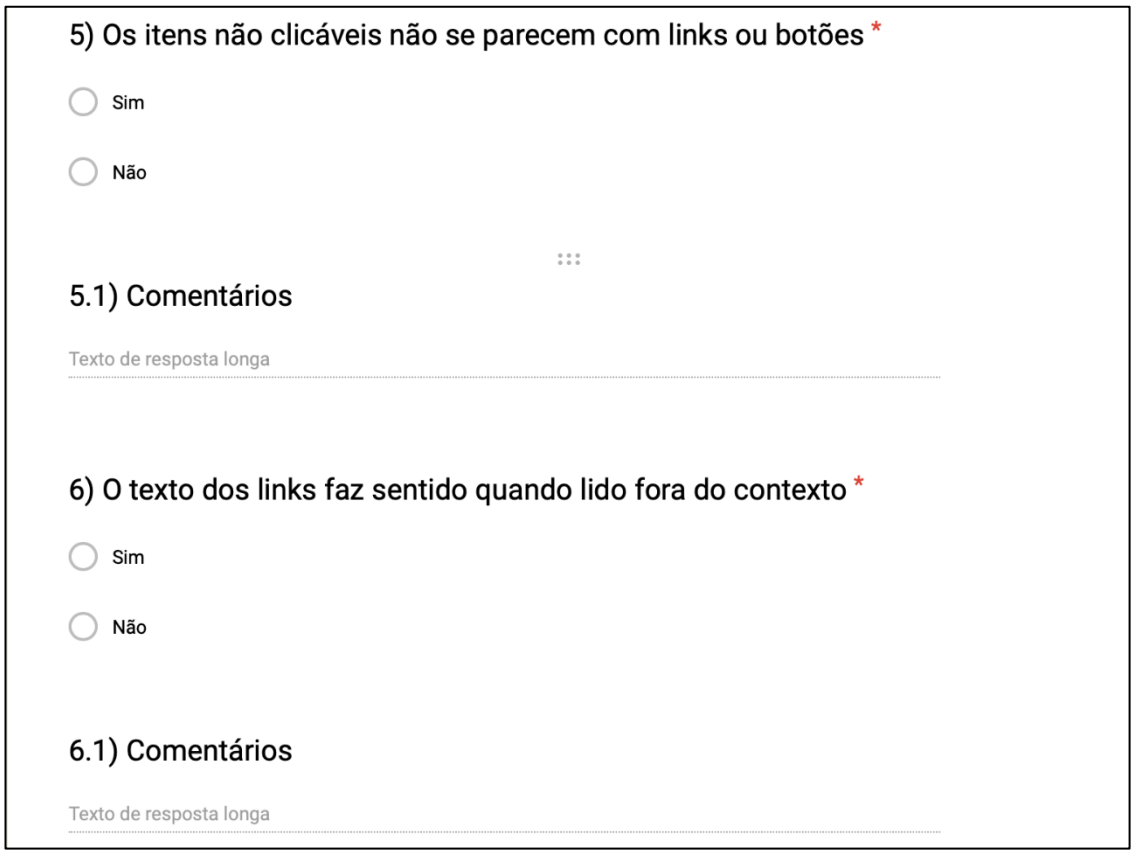
Fonte: elaborado pelo autor.

Figura 19 - Pergunta 3 e 4 do questionário



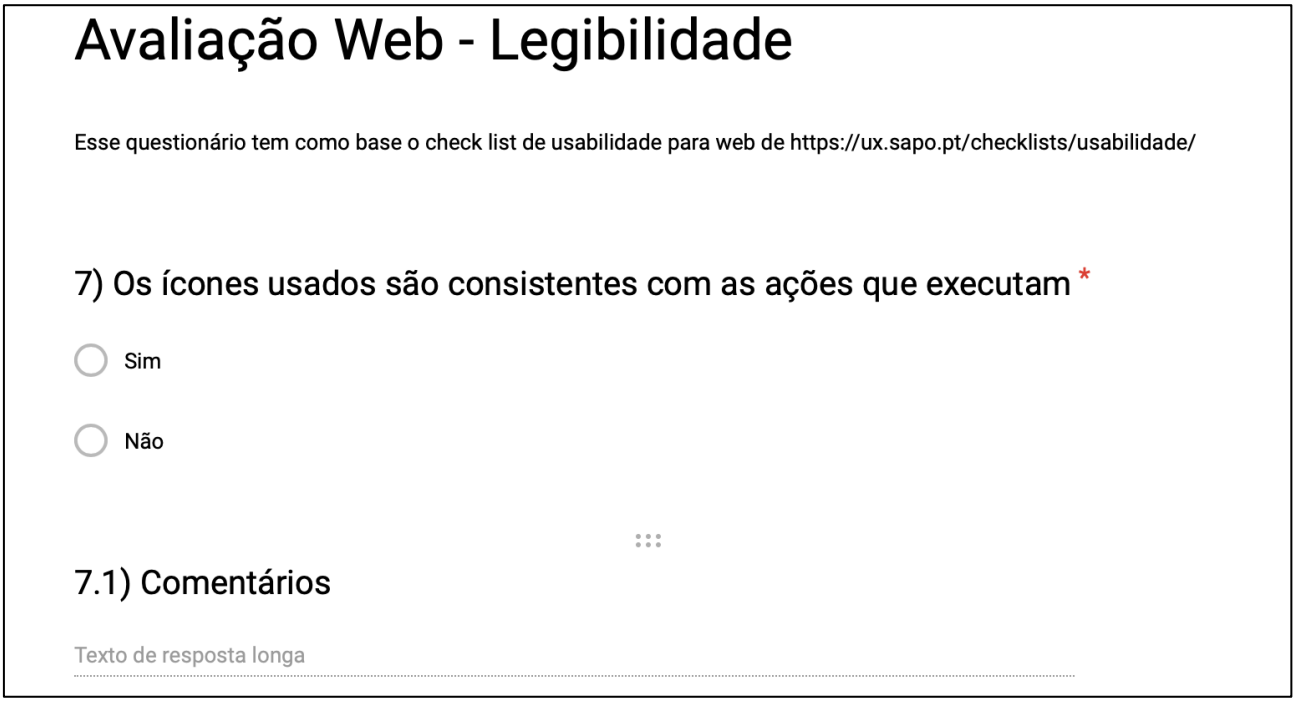
Fonte: elaborado pelo autor.

Figura 20 - Pergunta 5 e 6 do questionário



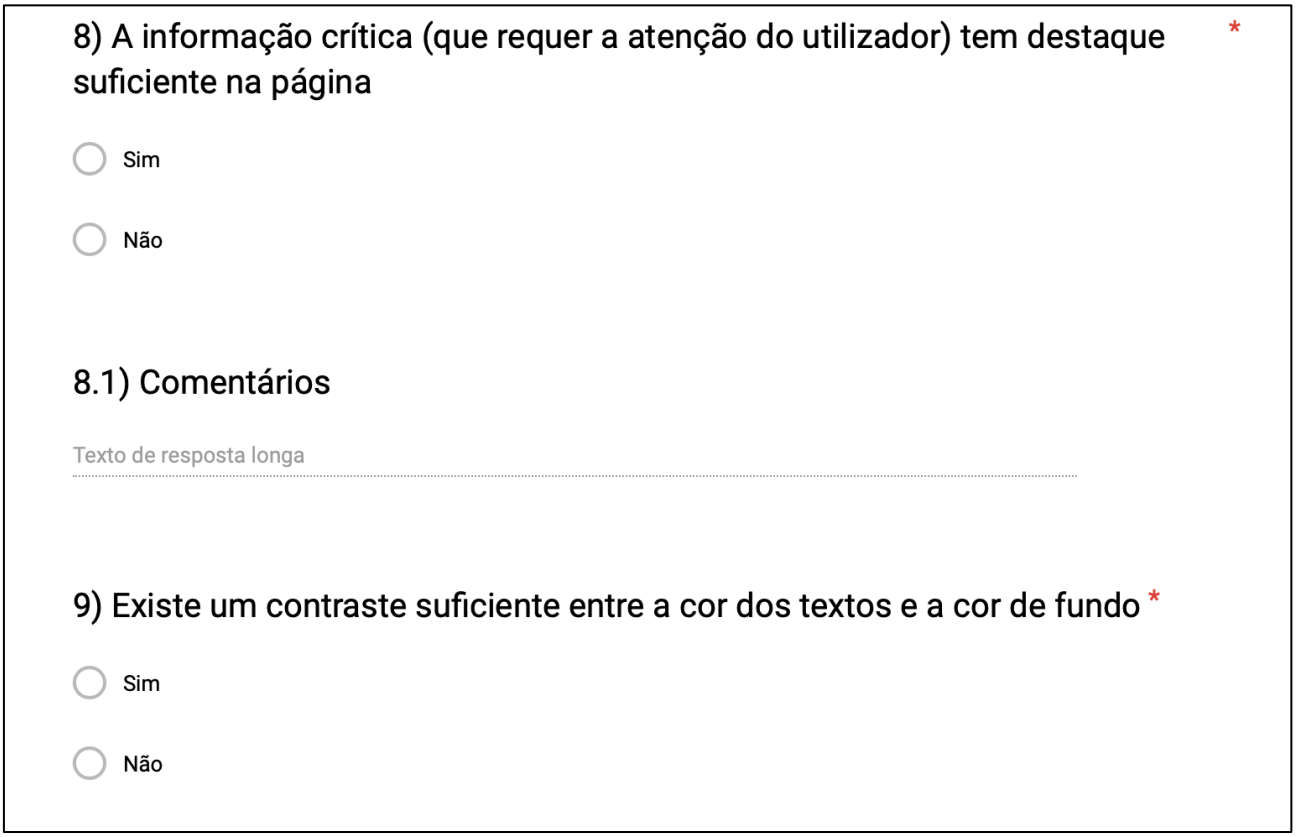
Fonte: elaborado pelo autor.

Figura 21 - Pergunta 7 do questionário



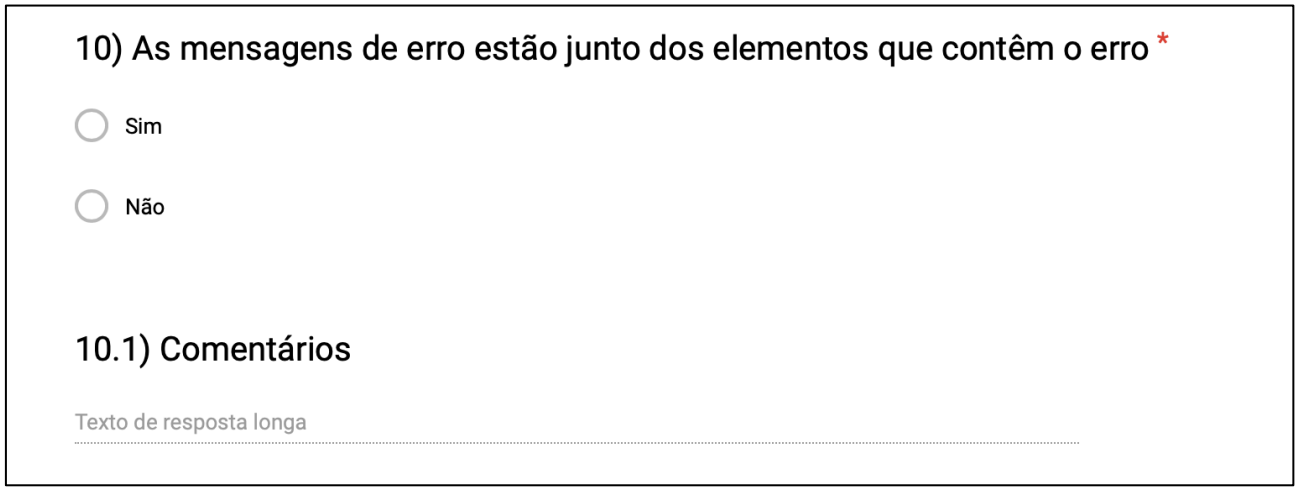
Fonte: elaborado pelo autor.

Figura 22 - Pergunta 8 e 9 do questionário



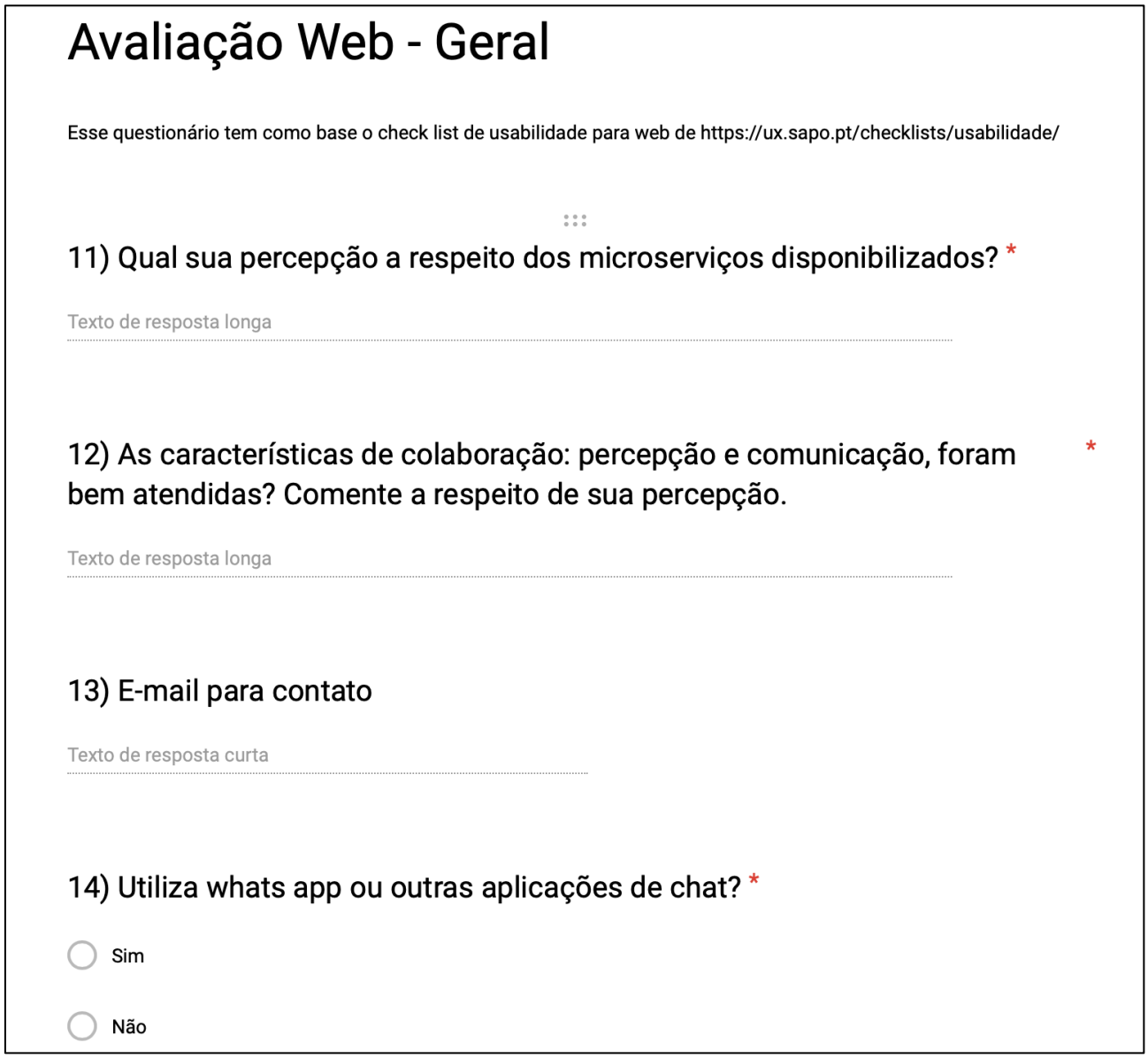
Fonte: elaborado pelo autor.

Figura 23 - Pergunta 7 do questionário



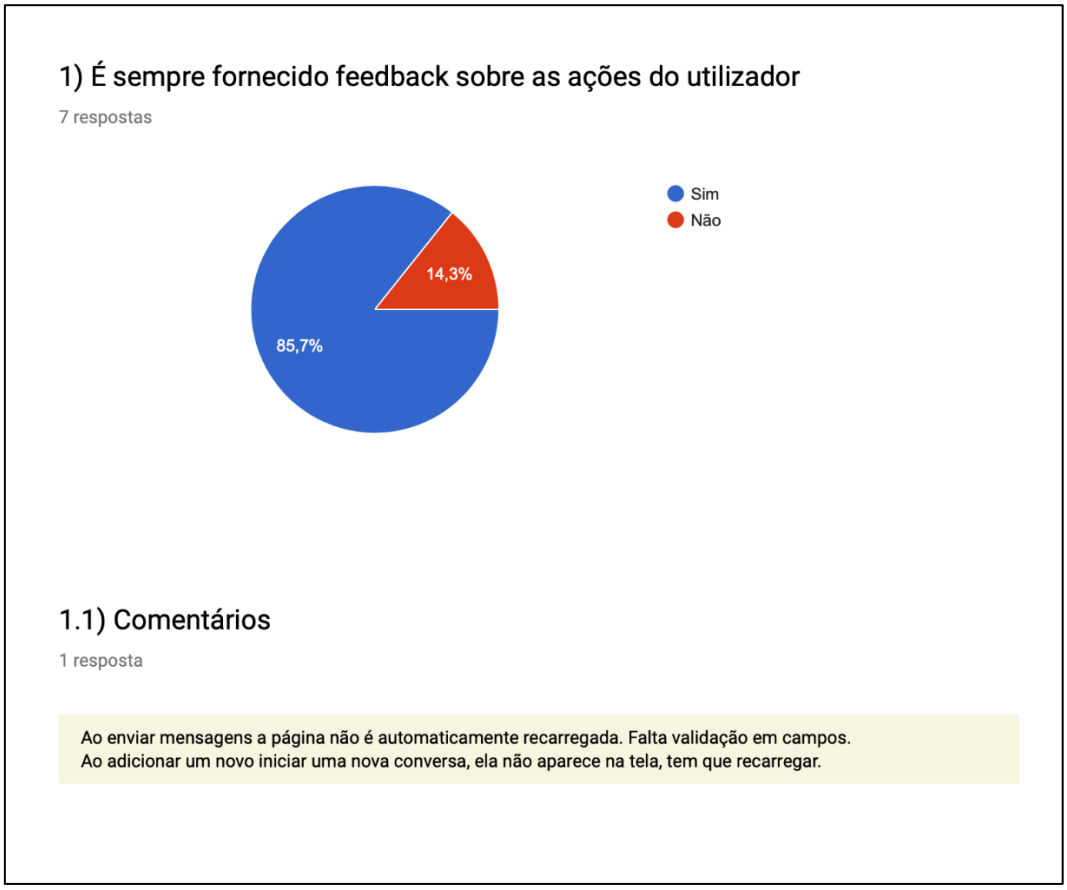
Fonte: elaborado pelo autor.

Figura 24 - Pergunta 11, 12, 13 e 14 do questionário



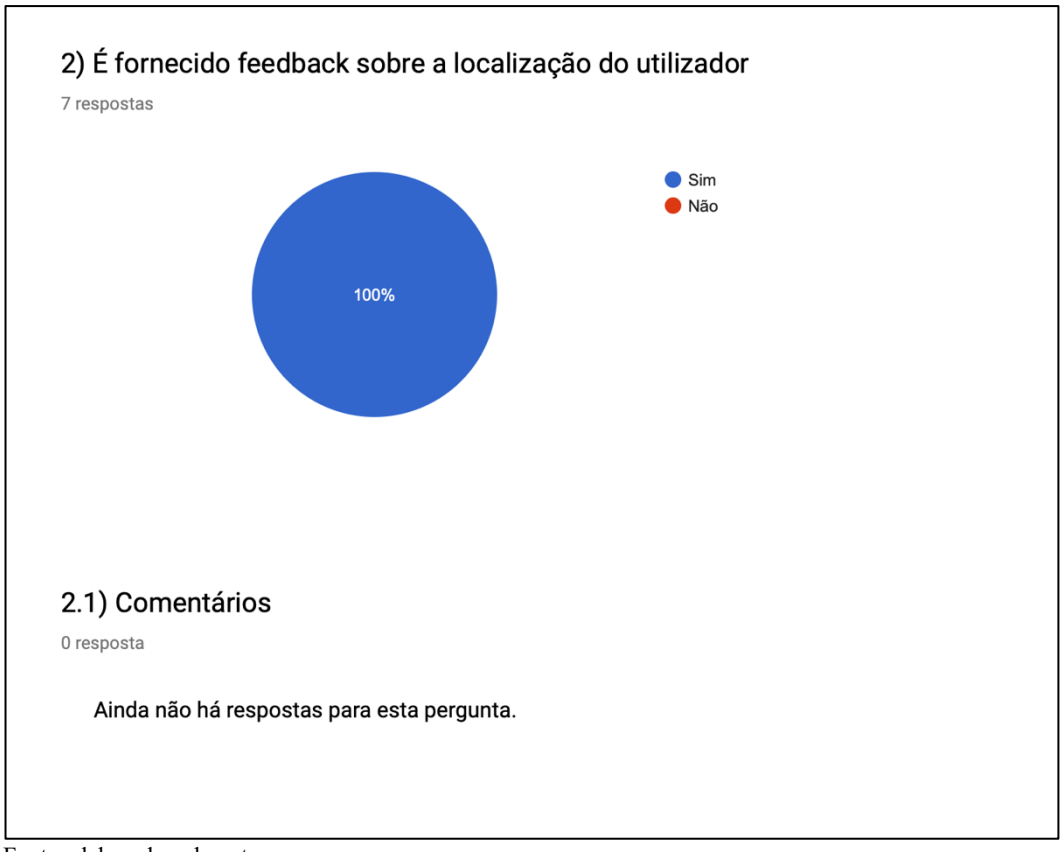
Fonte: elaborado pelo autor.

Figura 25 - Resposta da pergunta 1



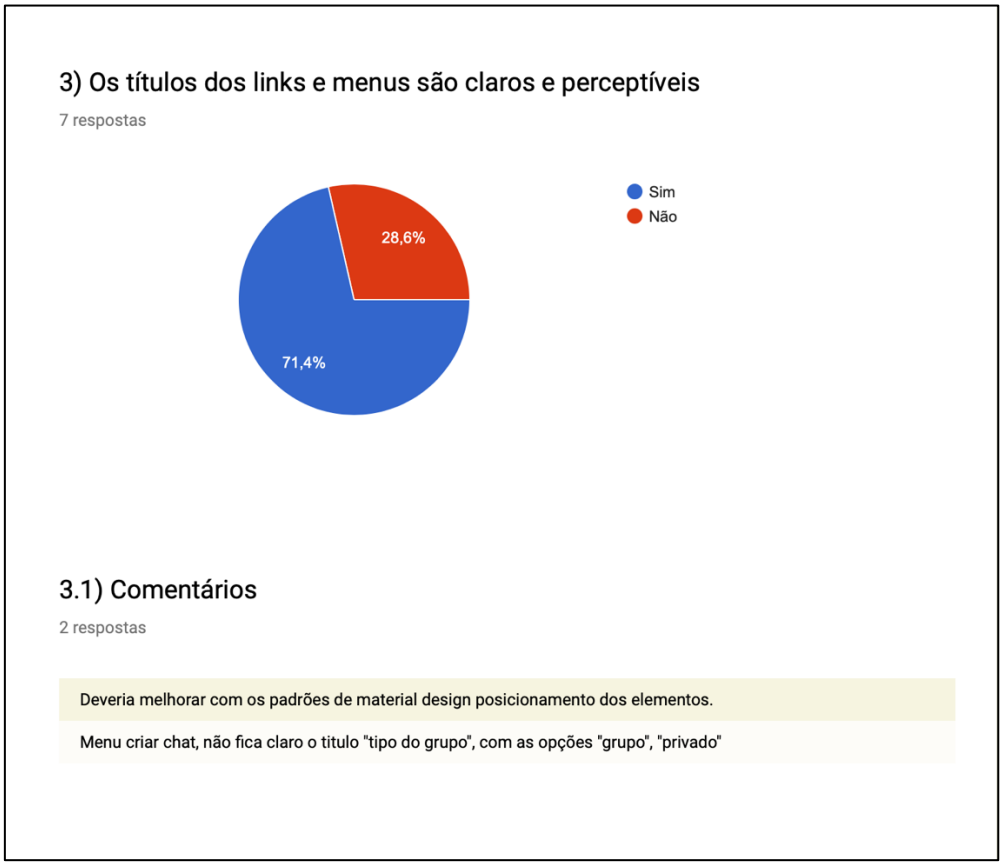
Fonte: elaborado pelo autor.

Figura 26 - Resposta da pergunta 2



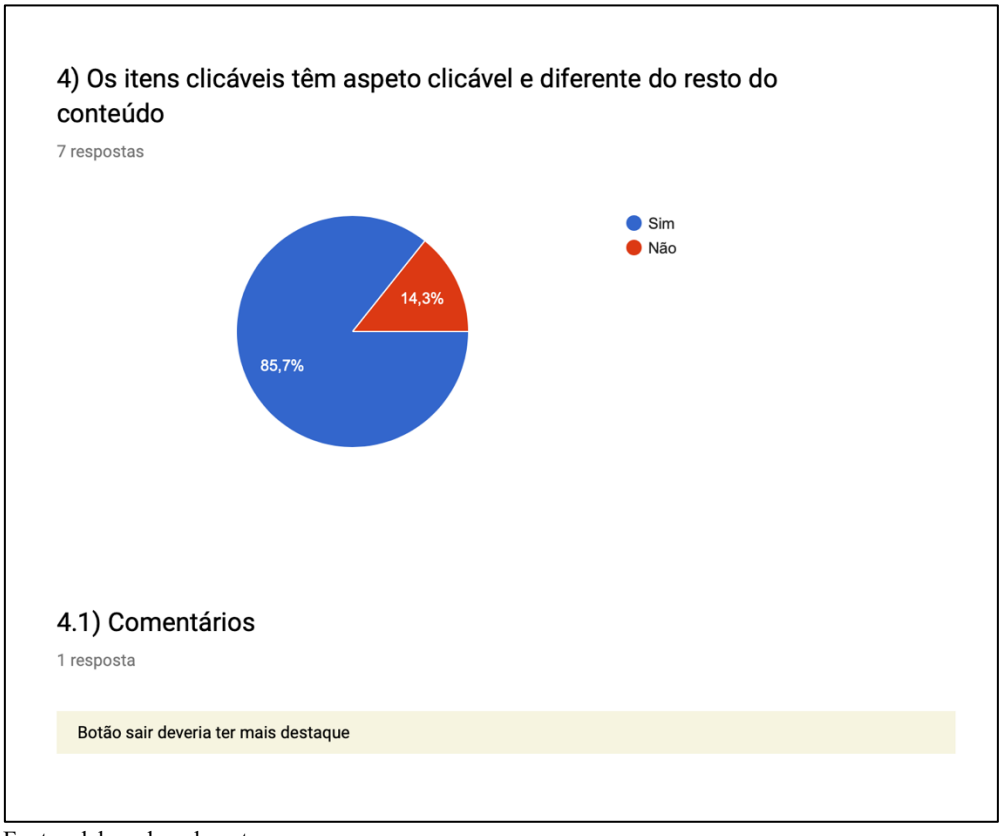
Fonte: elaborado pelo autor.

Figura 27 - Resposta da pergunta 3



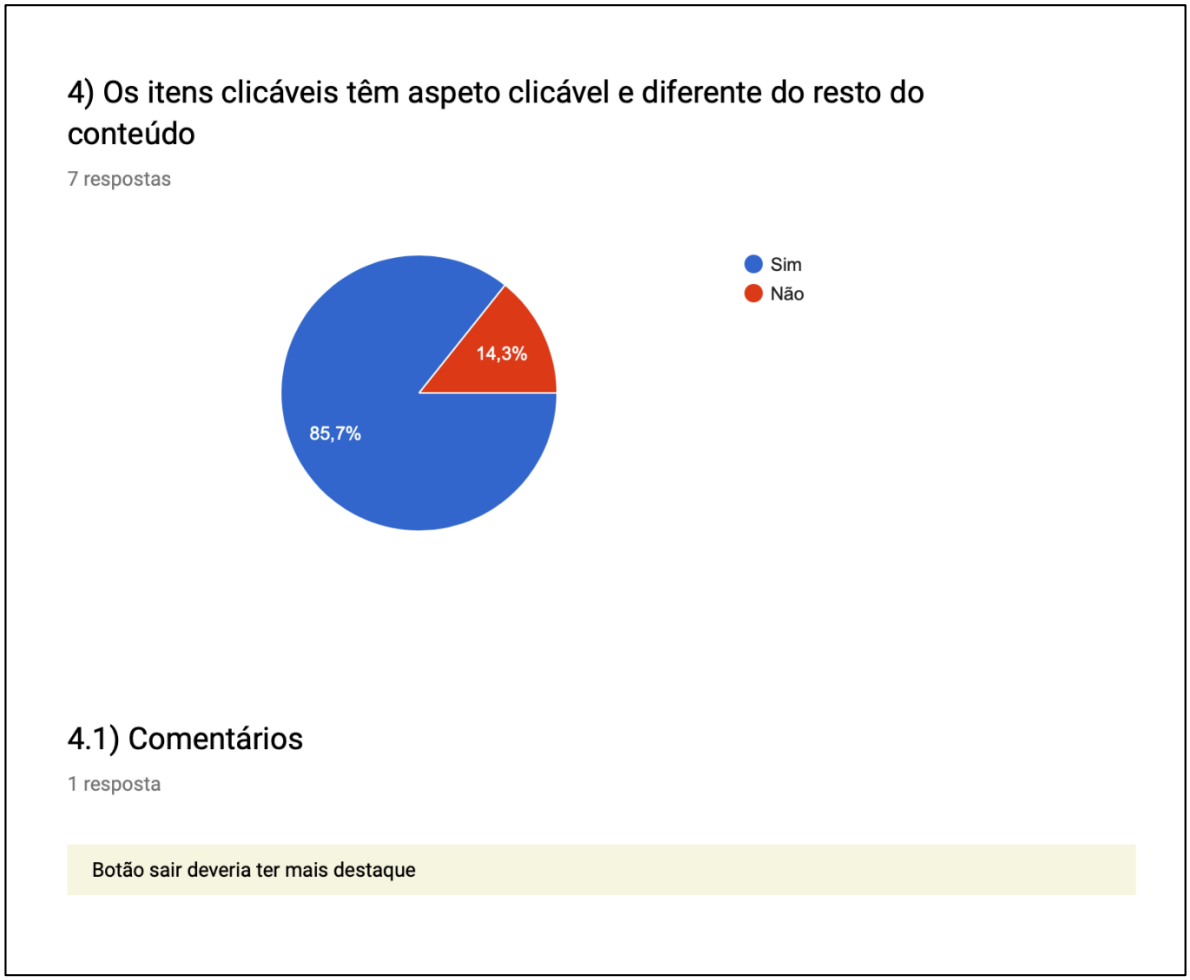
Fonte: elaborado pelo autor.

Figura 28 - Resposta da pergunta 4



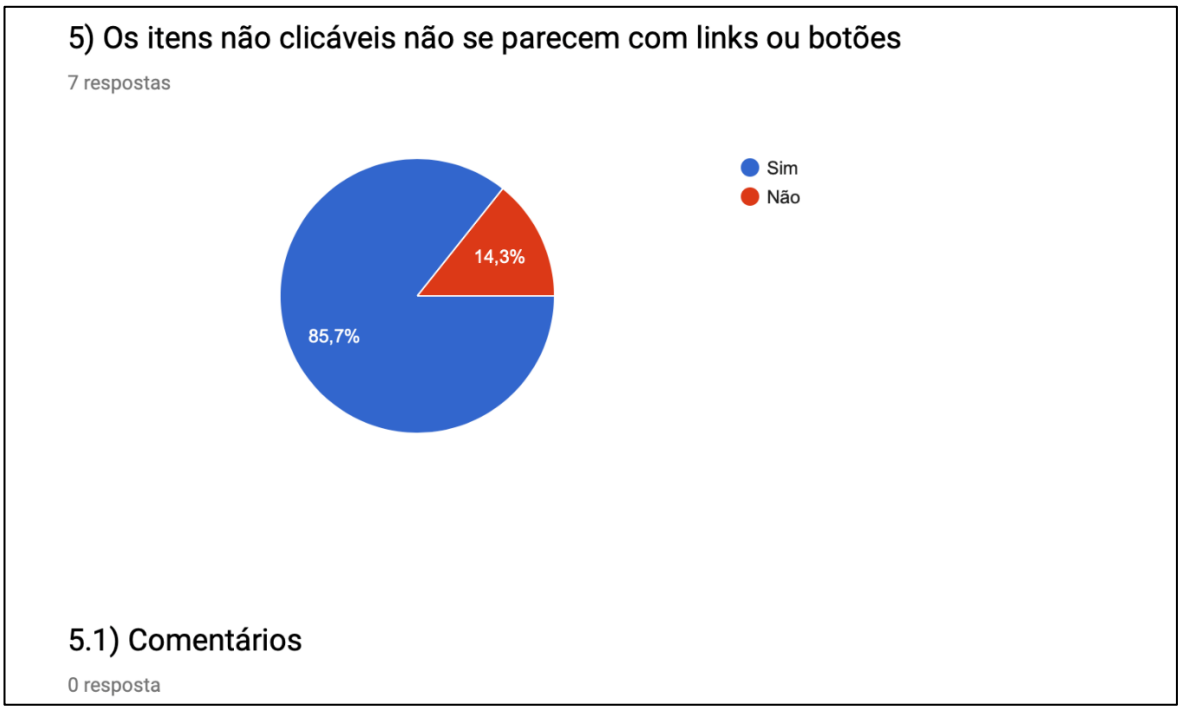
Fonte: elaborado pelo autor.

Figura 29 - Resposta da pergunta 4



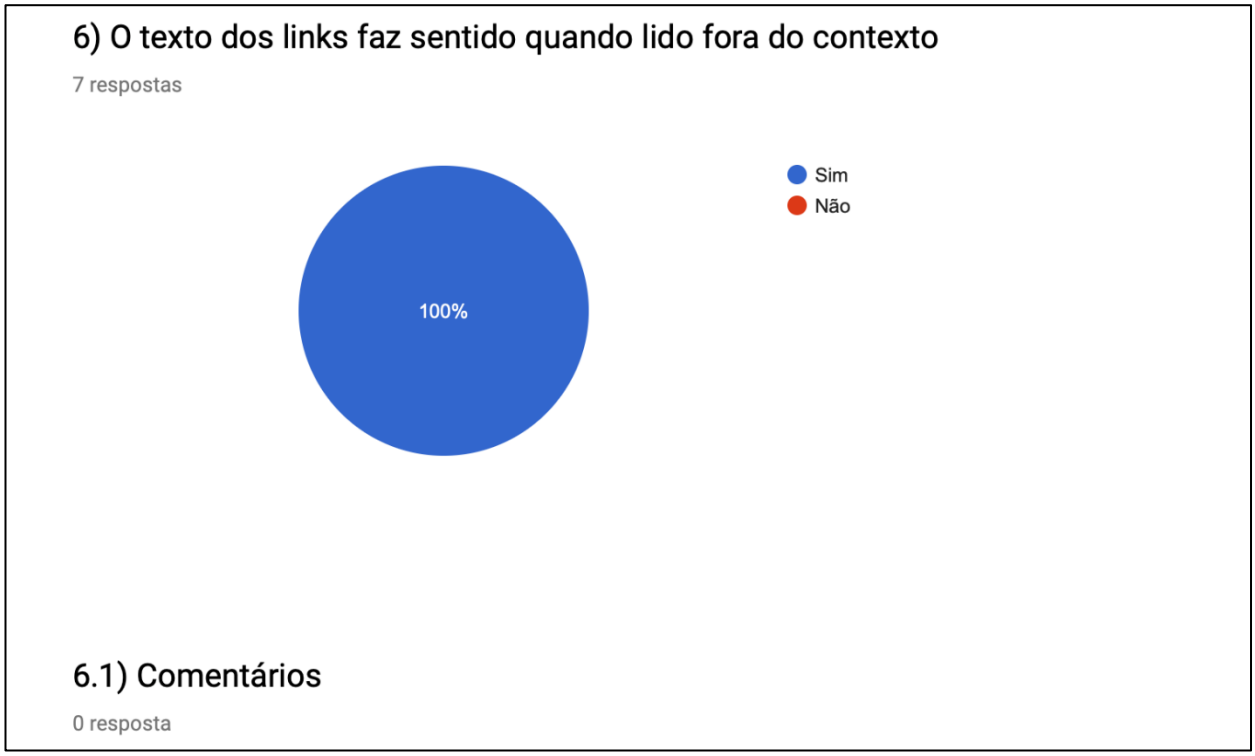
Fonte: elaborado pelo autor.

Figura 30 - Resposta da pergunta 5



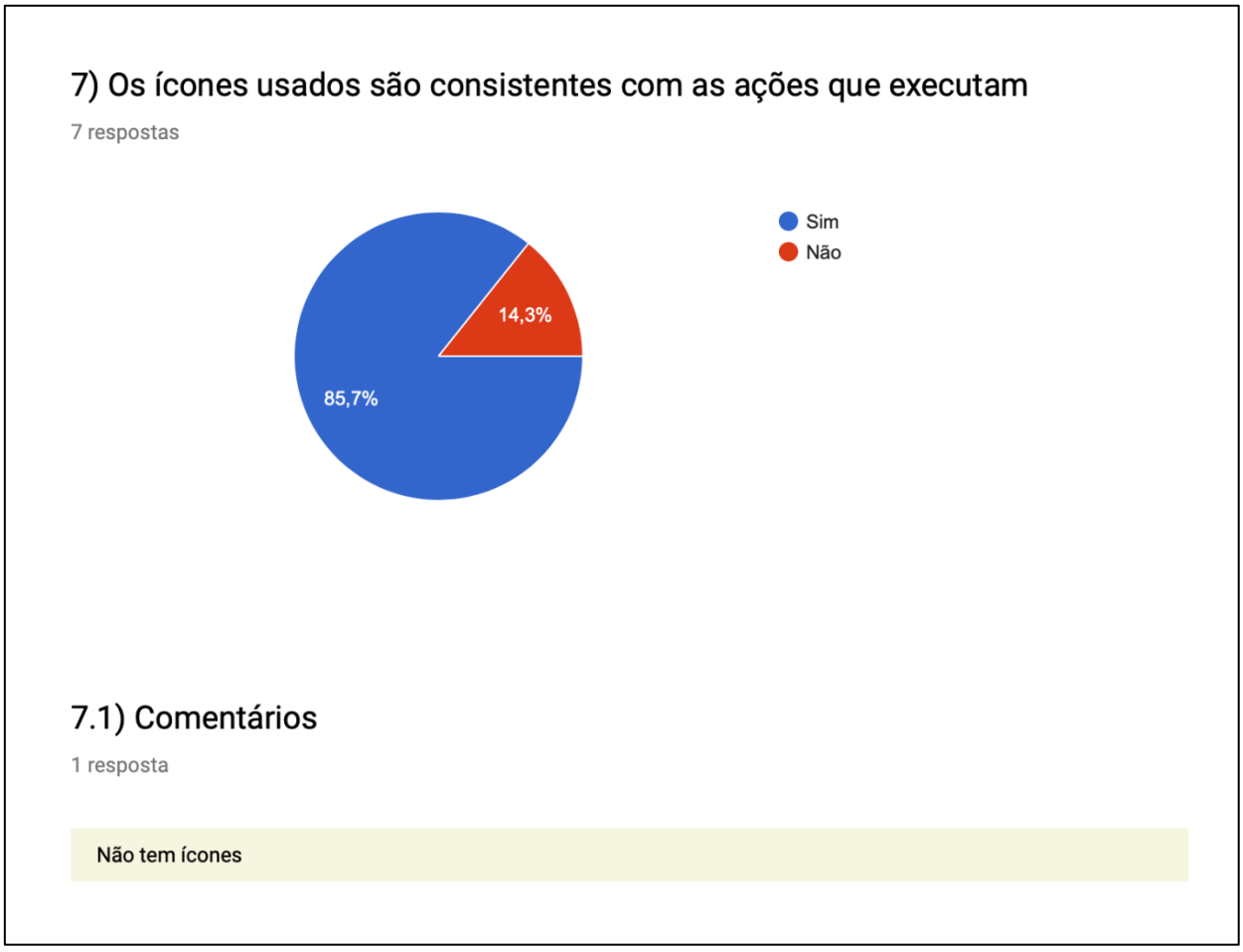
Fonte: elaborado pelo autor.

Figura 31 - Resposta da pergunta 6



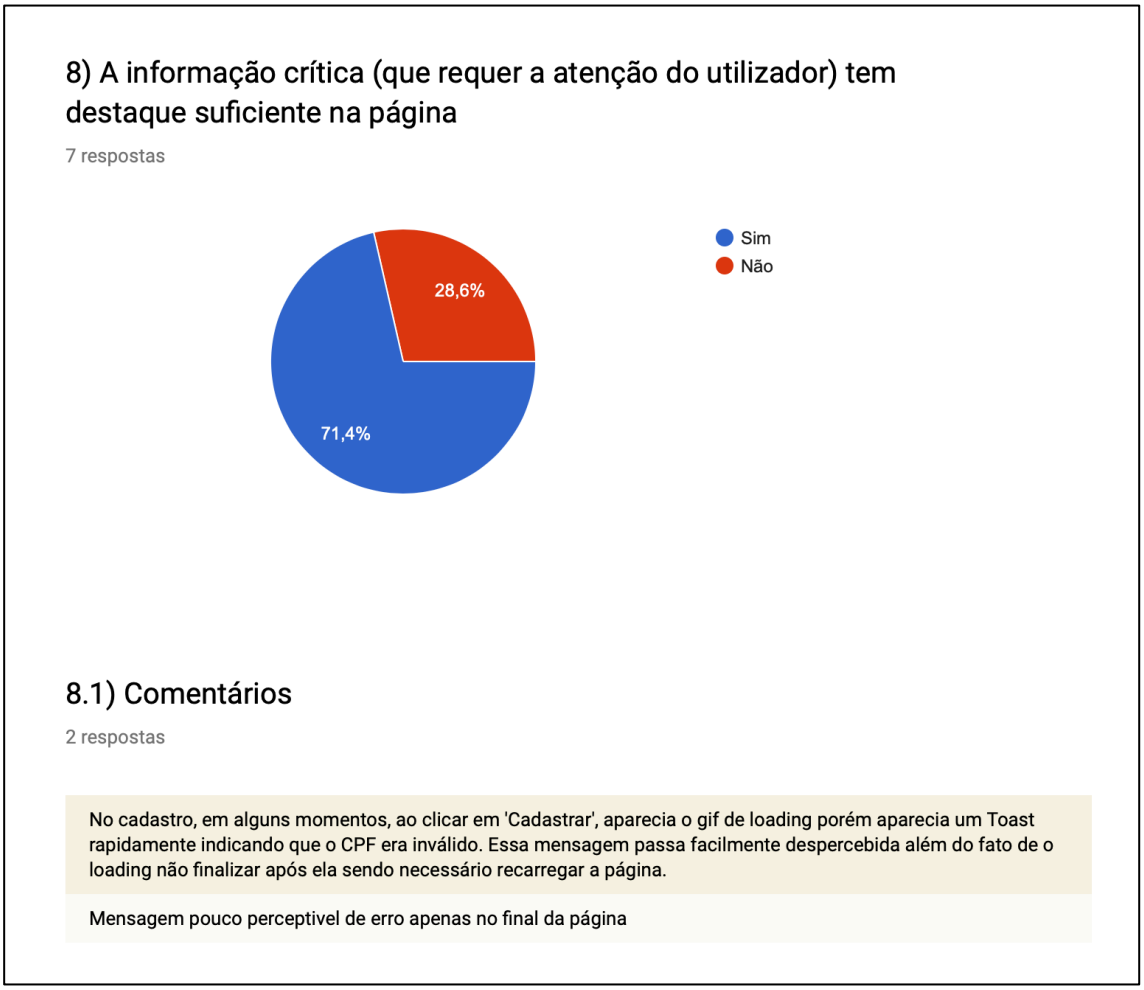
Fonte: elaborado pelo autor.

Figura 32 - Resposta da pergunta 7



Fonte: elaborado pelo autor.

Figura 33 - Resposta da pergunta 8



Fonte: elaborado pelo autor.

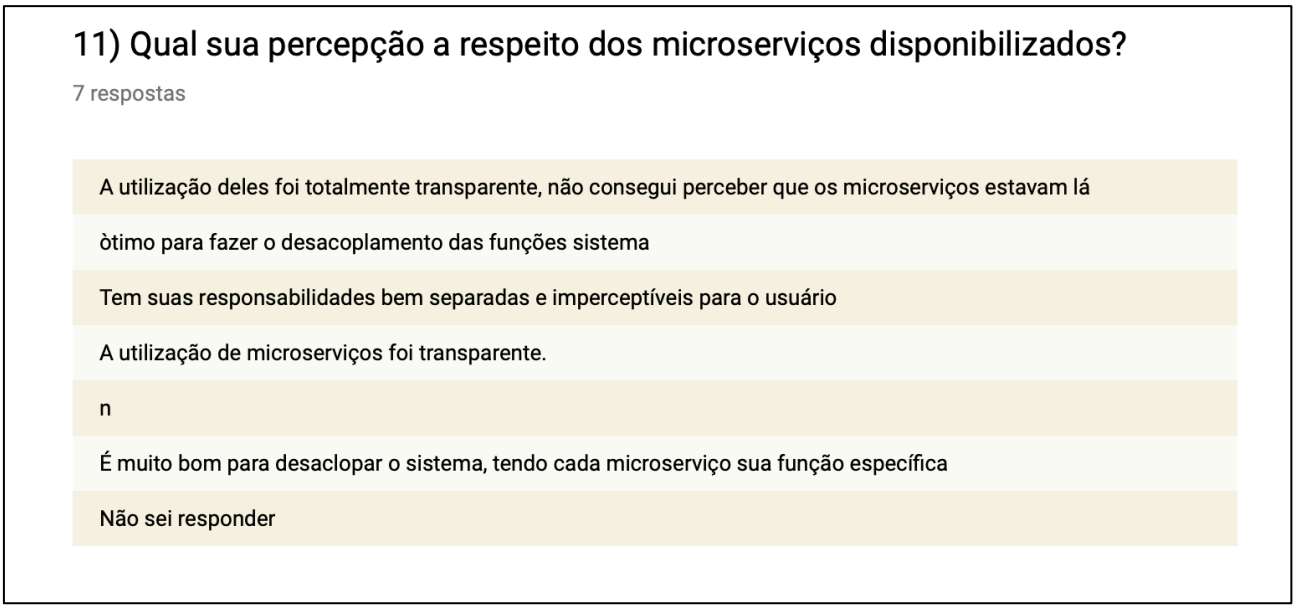
Figura 34 - Resposta da pergunta 9

Fonte: elaborado pelo autor.

Figura 35 - Resposta da pergunta 10

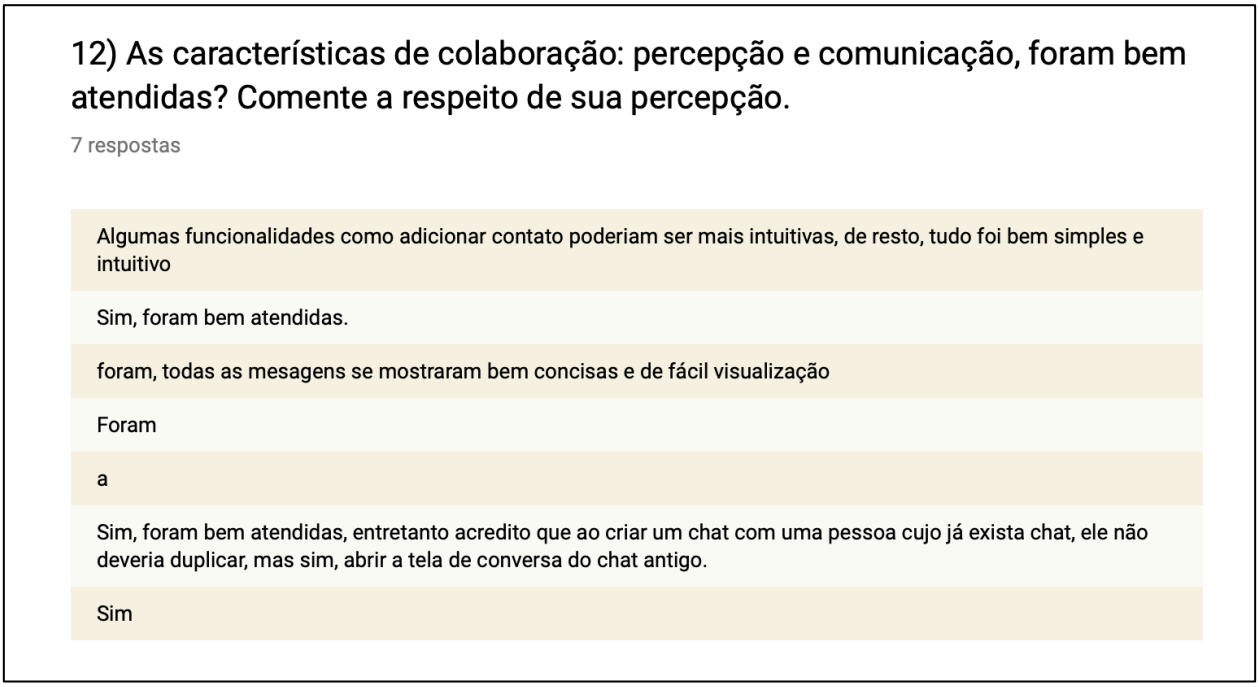
Fonte: elaborado pelo autor.

Figura 36 - Resposta da pergunta 11



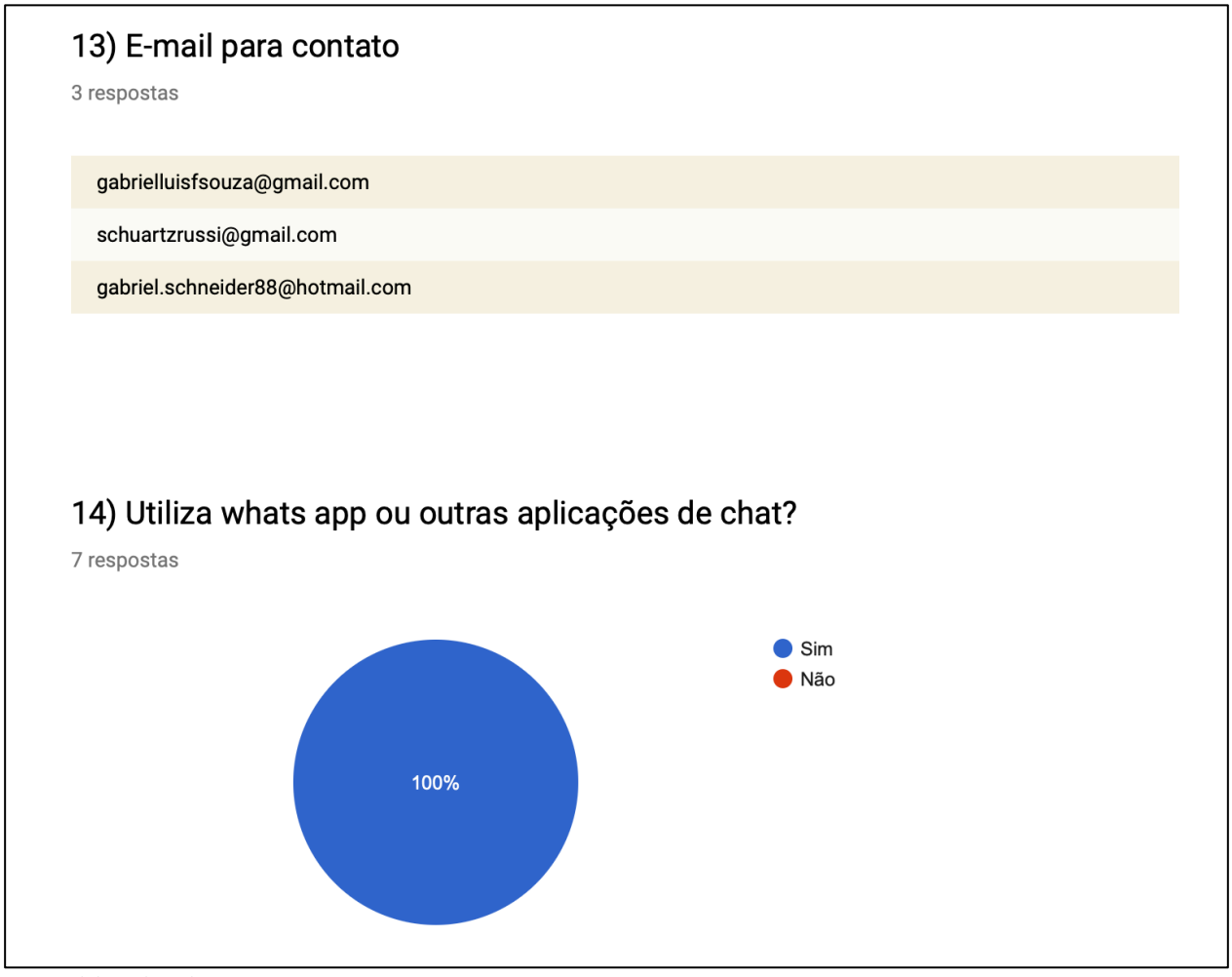
Fonte: elaborado pelo autor.

Figura 37 - Resposta da pergunta 12



Fonte: elaborado pelo autor.

Figura 38 - Resposta da pergunta 13 e 14



Fonte: elaborado pelo autor.