UNIVERSIDADE ESTADUAL DO TOCANTINS
CÂMPUS DE PALMAS
CURSO DE SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE APLICATIVO DE COMUNICAÇÃO BASEADO NA CRIAÇÃO DE SALAS DINÂMICAS CONFORME GEOLOCALIZAÇÃO DE USUÁRIOS

FRANCISCO VICTOR OLIVEIRA LUSTOSA

Palmas - TO

2022

UNIVERSIDADE ESTADUAL DO TOCANTINS
CÂMPUS DE PALMAS
CURSO DE SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE APLICATIVO DE COMUNICAÇÃO BASEADO NA CRIAÇÃO DE SALAS DINÂMICAS CONFORME GEOLOCALIZAÇÃO DE USUÁRIOS

FRANCISCO VICTOR OLIVEIRA LUSTOSA

Projeto apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

Palmas - TO
2022

CURSO DE SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE APLICATIVO DE COMUNICAÇÃO BASEADO NA CRIAÇÃO DE SALAS DINÂMICAS CONFORME GEOLOCALIZAÇÃO DE USUÁRIOS

FRANCISCO VICTOR OLIVEIRA LUSTOSA

Projeto apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

Me. Silvano Maneck Malfatti
Orientador

Professor
Examinador

Professor
Examinador

Palmas - TO

2022

Dados Internacionais de Catalogação na Publicação (CIP) Sistema de Bibliotecas da Universidade Estadual do Tocantins

L972d

LUSTOSA, Francisco Victor Oliveira
Desenvolvimento de aplicativo de comunicação baseado na criação de salas dinâmicas conforme geolocalização de usuários.. Francisco Victor Oliveira Lustosa. - Palmas, TO, 2022

Monografia Graduação - Universidade Estadual do Tocantins – Câmpus Universitário de Palmas - Curso de Sistemas de Informação, 2022.

Orientador: Silvano Maneck Malfatti

1. Estudo de caso da viabilidade de aplicativo de salas virtuais criadas a partir da geolocalização..

CDD 610.7

Elaborado pelo sistema de geração automatica de ficha catalográfica da UNITINS com os dados fornecidos pelo(a) autor(a).

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA FUNDAÇÃO UNIVERSIDADE ESTADUAL DO TOCANTINS - UNITINS

Aos **22** dias do mês de **Junho** de **2022**, reuniu-se na Fundação Universidade Estadual do Tocantins, Câmpus Palmas, Bloco B, às **08:30 horas**, sob a Coordenação do Professor **Silvano Malfatti**, a banca examinadora de Trabalho de Conclusão de Curso em Sistemas de Informação, composta pelos examinadores Professores **Silvano Malfatti** (Orientador), **Tamirys Virgulino Ribeiro Prado e Tayse Virgulino Ribeiro, Jocivan Suassone Alves** para avaliação da defesa do trabalho intitulado **"Desenvolvimento de Aplicativo de Comunicação Baseado na Criação de Salas Dinâmicas Conforme Geolocalização de Usuários"** do acadêmico **Francisco Victor Oliveira Lustosa** como requisito para aprovação na disciplina Trabalho de Conclusão de Curso (TCC). Após exposição do trabalho realizado pelo acadêmico e arguição pelos Examinadores da banca, em conformidade com o disposto no Regulamento de Trabalho de Conclusão de Curso em Sistemas de Informação, a banca atribuiu a pontuação: 9.5.

Sendo, portanto, o Acadêmico: (X) Aprovado ( ) Reprovado

Assinam esta Ata:

Professor Orientador: Silvano Maneck Malfatti

Examinador: Tamirys Virgulino Ribeiro Prado

Examinador: Tayse Virgulino Ribeiro

Examinador: Jocivan Suassone Alves

Prof. Silvano Malfatti

Presidente da Banca Examinadora
Coordenação do Curso de Sistemas de Informação

Este trabalho é dedicado à minha família, pelo apoio incansável e incondicional, o qual tornou possível a realização dessa etapa tão importante na minha vida.

Agradecimentos

Agradeço a Deus, em primeiro lugar, pela força, perseverança e oportunidade de concluir o curso. Aos meus pais e aos meus irmãos, que sempre acreditaram em mim mesmo em meio aos meus fracassos. Ao meu irmão, Ivan Borges de Lima (In memoriam), o qual sou grato e guardo infinitas saudades. Agradeço aos meus colegas que me acompanharam nesta jornada acadêmica e hoje seguem outros caminhos e aos meus professores que me foram exemplo e espelho, os quais lembrarei e guardarei com muito carinho pela importância que tiveram em minha vida, e ao meu orientador Me. Silvano Maneck Malfatti por me orientar no desenvolvimento desse trabalho.

"O grande mito do nosso tempo é que a tecnologia é comunicação".
Libby Larsen

Resumo

Com a difusão da internet, é possível comunicar-se com pessoas em qualquer lugar do mundo. Porém, nem todos conseguem se comunicar facilmente na vida real, interagir com pessoas desconhecidas e fazer novas amizades em qualquer lugar. As principais aplicações de bate-papo virtuais do Brasil atualmente necessitam de conexões a partir de informações pessoais previamente informadas ou de pessoas próximas. Esse trabalho apresenta um estudo de caso para desenvolvimento de um aplicativo de comunicação baseado na criação de salas dinâmicas conforme geolocalização de usuários. Sendo desenvolvido em uma arquitetura própria com o *front-end* implementado com o *framework Flutter* e um *back-end* em *Node.JS*.

Palavras-chaves: Bate-papos dinâmicos, Aplicativo de geolocalização, facilitação de comunicação, sala virtual por gelocalização.

Abstract

With the spread of the internet, it is possible to communicate with people anywhere in the world. However, not everyone can communicate easily in real life, interact with strangers and make new friends anywhere. The major virtual chat applications in Brazil currently require previously informed personal information or from close people. This paper presents a study of case for the development of a application communication based on the creation of dynamic rooms according to the location of users. The app was developed in a own architecture with the front-end implemented with the Flutter framework and back-end in Node.JS.

Key-words: Dynamics chats, geolocalization application, facilitation of communication, virtual room by geolocation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AOT - Ahead Of Time.

API - Application Programming Interface.

APP - Application.

HTTP - Hypertext Transfer Protocol.

IOT - Internet of Things.

IP - Internet Protocol.

JIT - Just In Time.

JSON - JavaScript Object Notation.

LGPD - Lei Geral de Proteção de Dados.

MVC - Model View Controller.

MVP - Model View Presenter.

MVVM - Model View ViewModel.

OSI - Open System Interconnection.

SGBD - Sistema Gerenciador de Banco de Dados.

TCP - Transmission Control Protocol.

UDP - User Datagram Protocol.

UI - User Interface.

UML - Unified Modeling Language.

VM - Virtual Machine.

WPF - Windows Presentation Foundation.

XAML - Extensible Application Markup Language.

Sumário

1 Introdução

De acordo com Holanda (2016), a internet revolucionou a cultura, economia e a comunicação mundial na década de 90 com a difusão da comunicação instantânea através de E-mails, bate-papos por computadores e chamadas de vídeo, dentro das limitações da época.

Desde então continuou a crescer, impulsionando quantidades cada vez maiores de informações on-line e entretenimento e redes sociais Holanda (2016), permitindo o acesso a informação e a comunicação virtual em tempo real a partir de qualquer lugar do mundo. Entretanto, as principais aplicações de bate-papo virtuais do Brasil, segundo Loureiro (2019) são elas: *Messenger, WhatsApp e Telegram*, atualmente são voltadas ao estabelecimento de conexões a partir de informações pessoais previamente informadas ou de pessoas próximas.

Essas aplicações não levam em consideração os usuários que estão no mesmo ambiente, se limitando a criação de amizades e relacionamentos a partir da iniciativa de alguém, dificultando a socialização de pessoas introspectivas, tímidas, inseguras ou com baixa autoestima.

Essa pesquisa tem como finalidade o estudo de uma nova abordagem de comunicação pública: Um estudo de caso de sistema de bate-papos em salas públicas virtuais criadas a partir da geolocalização de estabelecimentos, permitindo que somente as pessoas que estão no mesmo se comuniquem.

A aplicação funcionará encontrando estabelecimentos que estão em um raio de 5km do usuário, o permitindo visualizar o número de pessoas *online* e a distância, porém, só sendo possível acessar a sala virtual quando estiver na geolocalização do mesmo. Permitindo a possibilidade de solicitar uma sala privada com outro usuário específico, que terá a opção de aceitar ou não a solicitação. No momento que se deslocar para outra geolocalização, se estiver conectado, o usuário será desconectado. O estudo de caso verificará a viabilidade da aplicação do sistema de modo que possa facilitar a comunicação de pessoas desconhecidas que estejam na mesma localização.

1.1 Justificativa

Apesar das facilidades decorrentes da internet, a comunicação interpessoal em 2021 ainda possui dificuldades, dado que 80% dos jovens do mundo apresentam sintomas de timidez e insegurança Cury (2017). Mesmo com aplicações de bate-papo virtuais, não é possível estabelecer uma conexão direta sem previamente ter acesso a dados pessoais ou conhecidos já conectados, sendo necessário partir o interesse de algum lado, e nem todos possuem essa iniciativa.

Em buscas realizadas foram encontrados aplicativos que permitiam se conectar à pessoas desconhecidas que estejam no mesmo ambiente sem anteriormente possuir informações pessoais ou amigos próximos, porém, não seguindo o modelo de negócio apresentado neste. Portanto este trabalho pretende avaliar o estudo de caso de uma aplicação de comunicação entre usuários dinamicamente através da criação de salas virtuais geolocalizadas.

1.2 Objetivos

1.2.1 Objetivo Geral

Realizar um estudo de caso, baseado em referências acadêmicas, para desenvolver um sistema que permita a comunicação entre pessoas em ambientes públicos, a partir de salas virtuais públicas e privadas.

1.2.2 Objetivos Específicos

- Estudo de protocolos de comunicação *Unicast* e *Multicast*.
- Estudo e desenvolvimento de arquitetura de comunicação.
- Estudo e desenvolvimento de protocolo de aplicação.
- Estudo e desenvolvimento de mecanismo de controle de posicionamento por geolocalização.
- Publicação do sistema para teste.
- Realizar estudo de caso em ambiente público.

2 Referencial Teórico

O presente capítulo aborda os fundamentos teóricos e técnicos necessários para o desenvolvimento do projeto de conclusão de curso, abordando os aplicativos similares, protocolos de comunicação, embasamento teórico e as tecnologias necessárias para o desenvolvimento da aplicação.

2.1 Trabalhos Relacionados

Existem diversos trabalhos publicados que possuem relação com partes dos temas abordados nesse trabalho, como geolocalização e criação de salas de bate-papo virtuais. Como parte desses trabalhos encontrados pode ser citado o Moreira (2017) que consiste no desenvolvimento de um aplicativo web responsivo para dispositivos móveis com o intuito de criar uma plataforma para prestadores de serviço autônomos anunciarem seu trabalho para o mercado e facilitar um possível cliente a encontrar o serviço do mesmo, sendo utilizado a geolocalização para encontrar as pessoas próximas e uma página de bate-papo para negociações, possuindo similaridades com o trabalho aqui proposto aplicado a um tema diferente.

Campos (2015) Consistindo no desenvolvimento de um aplicativo multiplataforma que permitiria o compartilhamento de localização entre amigos e facilitaria a locomoção de visitantes em um evento "Interação FURB" que foi disponibilizado pela Universidade Regional de Blumenau. O aplicativo estaria integrado à rede social *Facebook*, possibilitando que os visitantes entrassem no aplicativo utilizando informações de seu perfil e enviasse convites para sua lista de amigos.

Gonçalves e Francisco (2020) Aborda um estudo de caso de um sistema para facilitar o transporte de estudantes e a compra de passagens utilizando recursos geolocalizados.

2.2 Aplicativos de Comunicação Existentes na Atualidade

Existem aplicativos móveis para trocar mensagens em *chats* com várias finalidades, entre elas estão criar amizades, encontrar parceiros para um relacionamento amoroso ou para manter um grupo de amigos mais próximo.

Sem dúvidas esses aplicativos mudaram a forma como a sociedade moderna se comunica e interage virtualmente possuindo propostas simples de, após um simples cadastro, seja por um pedido de conexão ou através de um dado pessoal, estabelecer um canal de comunicação entre duas ou mais pessoas. Entre esses aplicativos, os que mais se parecem

com a proposta desse projeto são: *WhatsApp, Instagram, Telegram* e *Tinder* (LOUREIRO, 2019).

2.2.1 WhatsApp

O *WhatsApp* é um dos aplicativos de *chats* mais usados no mundo para comunicações pessoais ou em grupo, possuindo 2 bilhões de usuários ativos e tendo mais de 100 Bilhões de mensagens trocadas por dia, segundo o site AFFDE (2021).

Segundo sua página oficial WhatsApp (2021), sua proposta é simples e direta: Trocar mensagens, fotos, arquivos, e vídeo-chamas e ligações entre pessoas utilizando como identificação o número de telefone, sendo obrigatório o cadastro do mesmo na plataforma, e se restringindo a uma conta por número.

As conversas podem acontecer a partir de qualquer lugar, sendo necessário somente a conexão com a internet, e as mensagens enviadas pelo aplicativo permanecem no dispositivo que as enviou.

Mesmo similar em alguns pontos, como ambos trocarem mensagens em tempo real, possui uma proposta diferente desse sistema, pois no *WhatsApp* as mensagens não são restringidas pela localização do usuário, nem destruídas assim que o mesmo se deslocar das coordenadas do estabelecimento.

2.2.2 Telegram

O Telegram possui uma proposta similar ao *WhatsApp*, sendo também um mensageiro em tempo real, e possuindo mais de 100 Bilhões de mensagens trocadas por dia, segundo o site Knoth (2021). Possuindo todos os recursos do WhatsApp e alguns próprios, sendo um dos seus maiores concorrentes. Porém, assim como no aplicativo abordado anteriormente, possui as mesmas diferenças em relação a esse sistema.

2.2.3 Instagram

Atualmente, o Instagram possui 1,22 bilhões de usuários ativos, sendo que 500 milhões acessam a plataforma todos os dias Gonçalves (2021), consistindo também em um mensageiro em tempo real e oferecendo recursos de conexão com pessoas a partir do nome.

Porém, mesmo possuindo semelhanças, a comunicação no aplicativo é estabelecida através de amizades ou do conhecimento previamente do nome de usuário. No caso da proposta dessa pesquisa a conexão seria feita sem a necessidade dessas informações.

2.2.4 Tinder

O Tinder possui uma proposta diferente dos aplicativos anteriormente citados, pois, segundo sua página oficial Group (2021), tem como foco oferecer um aplicativo de mensagens e vídeo chamadas para relacionamentos românticos ou casuais entre pessoas de qualquer lugar do mundo. Funcionando através de *matchs*[1] em pessoas próximas que possuem o aplicativo instalado, distância essa que é calculada de forma automática pelo mesmo.

Porém, mesmo possuindo um raio de aproximação de usuários, este sistema exige que os mesmos estejam próximos fisicamente, na mesma geolocalização.

2.2.5 Comparativo entre os Aplicativos

Tabela 1 – Tabela Comparativa Entre os Sistemas.

Funcionamento	WhatsApp	Instagram	Telegram	Tinder	Sistema Geolocalizado
Envio de mensagens em tempo real	X	X	X	X	X
Envio de arquivos	X	X	X		
Criação de Videochamadas	X	X	X	X	
Mensagens deletadas após deslocamento das coordenadas do estabelecimento					X
Necessidade de estar numa localização específica para trocar mensagens					X
Utilização de bots para interação com o estabelecimento					X
Necessidade prévia de identificação pessoal	X	X	X		
Utilização de geolocalização	X		X	X	X

Fonte: Autoria Própria (2021).

2.3 Protocolos de Rede

Kurose, Ross e Zucchi (2010) e Bungart (2017) afirmam que um protocolo tem como principal função interligar dois dispositivos, definindo o formato e ordem que os elementos serão entreguem, os permitindo se comunicar através de serviços de redes utilizados pelos usuários.

2.3.1 Arquitetura de protocolos de redes

Segundo Bungart (2017), a arquitetura de protocolos de redes se baseia no entendimento do modelo de referência *Open System Interconnection* (*OSI*) que possui camadas abstratas que diferenciam funções de uma comunicação.

Focando na correlação da arquitetura *TCP/IP* [2], com a camada *OSI*, a mesma agrupa as três camadas mais altas do sistema, aplicação, apresentação e sessão, em uma

[1] Match: Um sinal de interesse.
[2] TCP/IP: Protocolo TCP em conjunto com o protocolo IP.

só chamada de aplicação, que gerência as sessões, converte dados e desenvolve a aplicação em si Bungart (2017), englobando protocolos como *TCP*, *UDP*, *IP*, *HTTP*, entre outros (TEDESCO, 2019).

Segue abaixo uma figura representativa da correlação entre a camada e o protocolo.

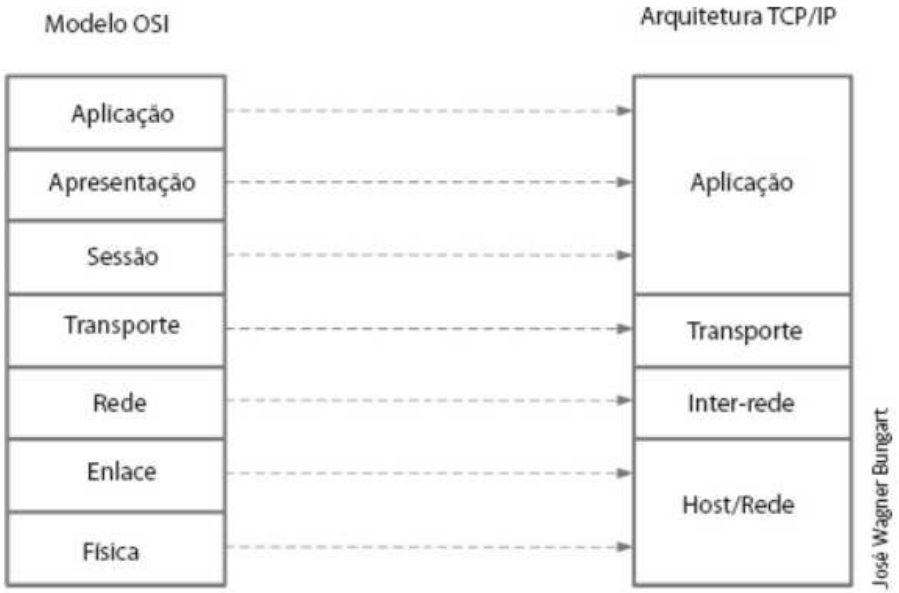

Figura 1 – Correlação OSI x TCP/IP.

Fonte: Bungart (2017) - p.68.

Na camada de transporte da arquitetura *TCP/IP* têm dois importantes tipos de protocolos de conexão utilizados para comunicação de dispositivos: o *Transmission Control Protocol* (*TCP*) e o *User Datagram Protocol* (*UDP*), "sendo o *TCP* orientado a conexão e o *UDP* não orientado a conexão" (FERREIA, 2021). Onde a orientação refere-se ao estabelecimento de uma conexão ao transmitir dados.

2.3.2 TCP x UDP

O protocolo *UDP* transmite dados através da camada de transporte, permitindo aplicações enviarem *datagramas*[3] encapsulados para o destinatário sem que seja necessário estabelecer uma conexão direta, possuindo um cabeçalho que registra duas portas que servem para identificar a máquina de origem e a de destino, e saber o que fazer com o pacote transmitido (TANENBAUM, 2003).

[3] Datagramas: Serviço não orientado a conexão, não possuindo confirmação de recebimento.

Tendo como característica não realizar controle de fluxo, congestionamento[4] e retransmissão após a recepção de um pacote incorreto e incompleto. Sendo comumente utilizado para conexões *Multicast*[5], e *Broadcast*[6].

Abaixo segue uma figura do cabeçalho completo de um protocolo *UDP*, possuindo além das duas portas, um campo de *comprimento* e um campo de *checksum*, utilizado para verificação da autenticidade do pacote que está sendo enviado.

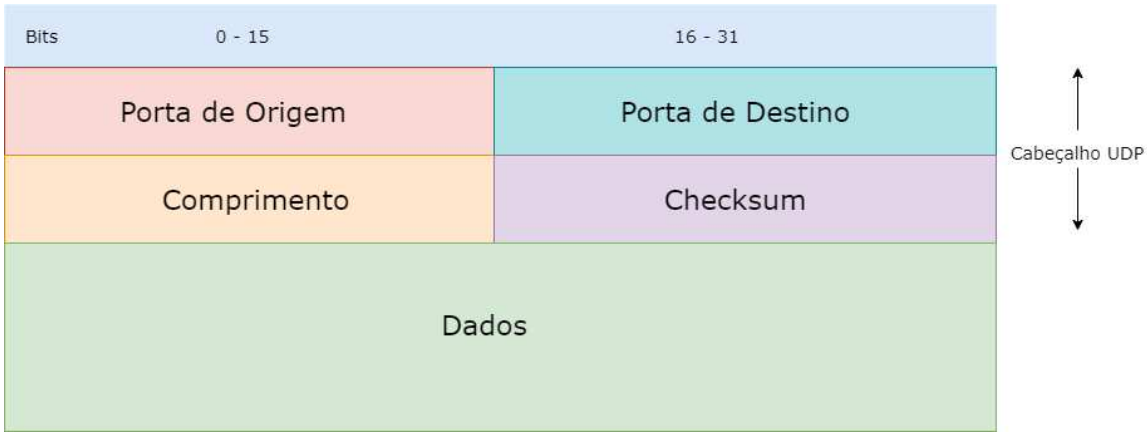

Figura 2 – Cabeçalho completo do Protocolo UDP.

Fonte: Reis (2016).

De acordo Tanenbaum (2003), o protocolo *TCP* foi desenvolvido para oferecer um fluxo de dados fim a fim confiável, operando em uma rede interligada. Sendo projetado para se adaptar às propriedades da rede interligada, sendo mais resiliente diante de possíveis falhas que possam vir a acontecer. Além de oferecer segurança, possui topologias, larguras de banda, atrasos, tamanhos de pacotes, entre outros parâmetros completamente diferentes da rede interligada.

Também possui um cabeçalho que registra duas portas que servem para identificar a máquina de origem e a de destino, sendo adicionado mais opções como número de sequenciamento, número de confirmação, comprimento, e outras informações necessárias para verificação e controle de fluxo de pacotes.

[4] congestionamento: Limitando ou aumentando a taxa de entrega de dados.
[5] Multicast: Para vários destinatários em um grupo especifíco de IPs.
[6] Broadcast: Para todos os destinatários conectados a rede.

Tendo como característica permitir uma conexão confiável e *full-duplex*[7], sequenciada, com controle de congestionamento, fluxo, e retransmissão de pacotes identificados como incompletos. Sendo comumente utilizado para conexões *Unicast*[8] (TANENBAUM, 2003).

Segue abaixo uma figura do cabeçalho completo de um protocolo *TCP*.

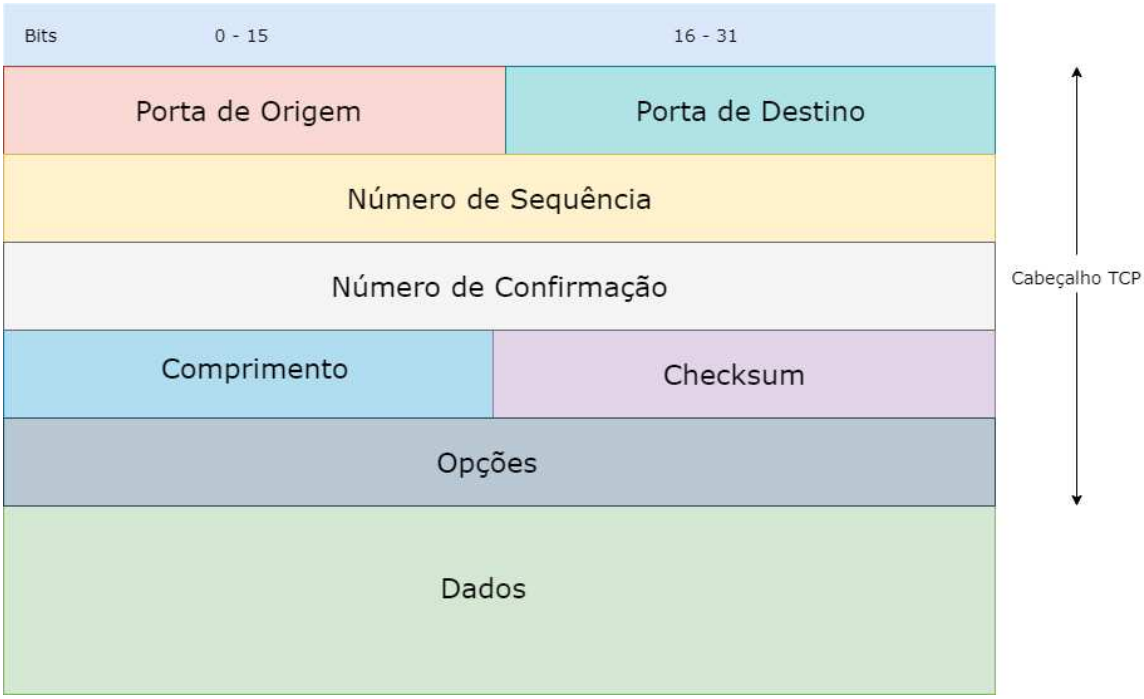

Figura 3 – Cabeçalho completo do Protocolo TCP.

Fonte: Reis (2015).

2.3.3 Soquetes

O soquete é a ligação entre um receptor e um transmissor a partir de um serviço *TCP/IP*, possuindo uma porta específica como canal para o fluxo de informações, associado ao protocolo *Hyper Text Transfer Protocol* (*HTTP*) utilizado para estabelecer a conexão. Um mesmo soquete pode receber várias conexões ao mesmo tempo (TANENBAUM, 2003).

[7] Full-duplex: Troca de informações de ambos os lados ao mesmo tempo.
[8] Unicast: Conexão estabelecida com um destinatário especifico.

Segue abaixo uma figura ilustrativa da conexão via soquete entre cliente-servidor.

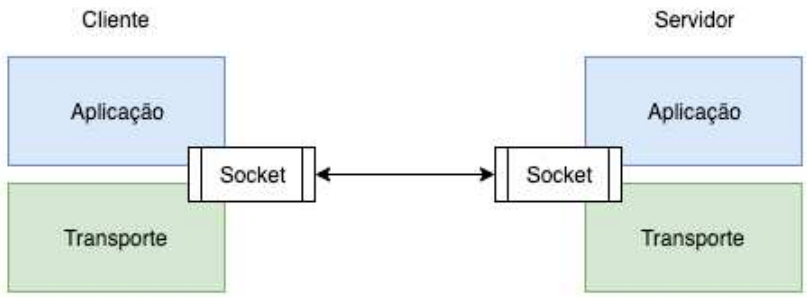

Figura 4 – Conexão entre Cliente e Servidor.

Fonte: Tedesco (2019).

No sistema serão utilizados soquetes em *multicast* (*Datagram Socket*[9]) para troca de mensagens entre os integrantes das salas públicas, onde todos que estiverem no grupo de IPs referentes àquela sala poderão trocar mensagens, sendo aplicado técnicas de verificação de integridade das informações recebidas e sequenciação de forma externa para reduzir significativamente as perdas de pacotes.

E soquetes em *unicast* (*Stream Socket*[10]) para troca de mensagens entre os integrantes de uma sala privada.

[9] Datagram Socket: Soquetes que não terão confirmação de entrega para os usuários.
[10] Stream Socket: Soquetes que possuem características de uma conexão TCP.

2.4 Dart e Flutter 2 para o Front-End

A escolhida da linguagem Dart para o projeto foi parte por sua semelhança sintática com linguagens de programação já existentes no curso de Sistemas de Informação da UNITINS, como *Java*, *JavaScript* e *C#*, reduzindo assim a curva de aprendizado, como por ser a linguagem de programação do *framework*[11] *Flutter*, o qual será usado para desenvolvimento da aplicação *front-end*[12], atendendo as necessidades exigidas para implementar o sistema, e por ser a linguagem nativa do sistema operacional recém lançado da *Google*, chamado *Fuchsia*.

O *Fuchsia* foi feito do zero para substituir o *Android*[13], pois apesar de ser rápido, ainda possui limitações de modificações de controle de código à nível de *hardware*[14], devido ao seu *microkernel* ser o *linux*.

Diferente do *Android*, é construído sobre um *microkernel* chamado *Zircon*, composto por conjunto de serviços, *drivers* e bibliotecas para que o *hardware* possa interagir diretamente com o *software*[15], resultando em uma grande melhoria de desempenho, controle sobre as instruções executadas e compatibilidade com *hardwares* mais simples (ANDRADE, 2020b).

Permitindo que o *Fuchsia* possua futuramente integração com dispositivos de *IoT*[16], sendo embarcado em carros, geladeiras, celulares, e todos os dispositivos que tenham suporte a sistema operacional. Além disso, os aplicativos desenvolvidos para o mesmo serão feitos nativamente em *Flutter* (ANDRADE, 2020b).

O *Dart* é *Ahead Of Time*(*AOT*) e *Just In Time*(*JIT*) trazendo a perfomance da compilação através da *Dart Virtual Machine* (*Dart VM*) e a velocidade da interpretação de ciclos de desenvolvimento sem a necessidade recompilação de todo o código Oficial (2011), além de trabalhar de forma excelente com objetos em notação *JSON*, sendo exatamente o que é necessário para a implementação desse projeto (MARINHO, 2020).

E, o *Flutter* é *framework* de desenvolvimento de código aberto escrito em *Dart* criado pela *Google*, sendo usado para desenvolvimento de aplicações híbridas (*Android* e *iOS*), *Web* e *Desktop* (OFICIAL, 2017).

[11] Framework: Estrutura de códigos e funções prontas que auxiliam o desenvolvimento.
[12] Front-End: Aplicação do lado do cliente.
[13] Android: Atual sistema operacional que a mesma utiliza em seus dispositivos.
[14] Hardware: Sistemas microeletrônicos.
[15] Software: Sequência de instruções que serão executadas pelo sistemas microeletrônicos.
[16] IoT: Ligação de objetos do nosso cotidiano com a internet por meio de um sistema operacional.

Com apenas o desenvolvimento de um código, o mesmo pode ser compilado para a plataforma escolhida onde utilizará os componentes nativos do sistema, obtendo um ótimo desempenho, enquanto mantém o *layout* nativo. Diferente de *frameworks* concorrentes como o *React Native*, que utiliza *"bridges"*[17] para acessar os recursos nativos e compilar para diferentes sistemas operacionais (ANDRADE, 2020a).

Abaixo pode ser visto a comparação entre as formas de acesso à recursos pelo *Flutter* e *React Native*.

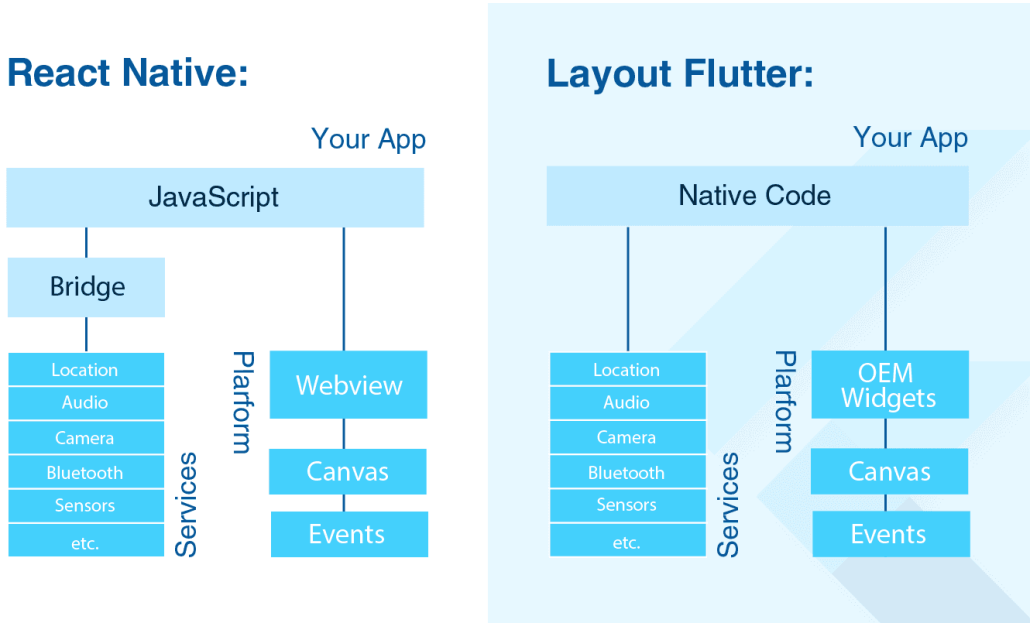

Figura 5 – Flutter x React.

Fonte: Google Imagens.

Tendo como particularidade sua compilação para *iOS*, sistema para dispositivos móveis da *Apple*, sendo possível somente com um sistema *MacOS*.

Atendendo de forma assertiva o objetivo da aplicação que é possuir uma versão móvel híbrida nativa para todos os usuários, e uma versão *Web* para ser utilizada somente pelos representantes das empresas, com apenas um único código e bom desempenho em dispositivos de baixo custo.

[17] Bridges: bibliotecas de terceiros ou recursos adicionais.

2.5 TypeScript e Node.JS para o Back-End

O *TypeScript* é uma linguagem de programação interpretada desenvolvida pela *Microsoft* a partir da linguagem de programação *JavaScript*, criada em meados da década de 90 pela *Netscape Communications*. Implementando recursos da Programação Orientada a Objetos à linguagem *JavaScript* (CAVALCANTE, 2021).

Sendo escolhido pela sua versatilidade, interpretação em tempo de execução, leveza e rapidez.

O *back-end* foi implementado utilizando o *framework Node.Js*, robusto e escalonável, sendo utilizado para gerenciar as salas públicas criadas a partir de um raio de 5km entre o usuário e os estabelecimentos, salas privadas criadas a partir da solicitação de usuários, e notificação de mensagens das mesmas.

3 Metodologia

A metodologia abordada nesse trabalho é a pesquisa aplicada e bibliográfica, a pesquisa aplicada refere-se ao desenvolvimento, implantação de um sistema de criação de salas dinâmicas a partir de geolocalização dos estabelecimentos, enquanto a pesquisa bibliográfica, como abordado por Fonseca no livro "Apostila de metodologia da pesquisa científica", refere-se ao embasamento teórico necessário para o desenvolvimento e integração do sistema, como artigos científicos, livros, monografias, dissertações, e páginas de *web* (FONSECA, 2002).

3.1 Elicitação de requisitos e definição do escopo do Sistema

Através de reuniões com o orientador do projeto, foi definido o escopo do projeto, regras de negócio, funcionalidades, requisitos funcionais e não-funcionais, telas, e o comportamento do sistema para se adequar ao objetivo geral do trabalho.

3.2 Materiais

Para desenvolvimento do sistema foram utilizadas as seguintes linguagens de programação:
Linguagem *Dart* aplicada ao *framework Flutter* para o desenvolvimento do *front-end* e a linguagem *JavaScript* (com o "*superset" Typescript*) aplicada ao *framework Node.Js* para desenvolvimento do *back-end*.

Sendo para o *front-end*:

- *Shared Preferences*: É uma solução para persistência de dados na memória interna do *smartphone* (FLUTTER.DEV, 2022).
- *Socket.IO Client*: consiste em uma biblioteca que permite a comunicação em tempo real, bidirecional e baseada em eventos entre o emissor e servidor, sendo aplicada sua versão para cliente no *front-end* (SOCKET.IO, 2014).
- *Bloc*: Biblioteca para implementar padrão de projeto para gerenciamento de estado de forma simplificada (BLOCLIBRARY.DEV, 2018).
- *Geolocator*: Biblioteca para implementar geolocalização forma simplificada (BASEFLOW.COM, 2018).

- *Url Launcher*: Biblioteca para implementar o redirecionamento de sites a partir do próprio aplicativo de forma simplificada (FLUTTER.DEV, 2019).

E no *back-end*:

- *Socket.IO*: consiste em uma biblioteca que permite a comunicação em tempo real, bidirecional e baseada em eventos entre o emissor e servidor, sendo aplicada sua versão para *back-end* (SOCKET.IO, 2014).

Tendo a partir do livros Marinho (2020) e Araújo (2021a), aprendido as possibilidades do *framework* e uma introdução a linguagem *Dart*, criação de projetos, técnicas de comunicação com agentes externos, padrões de projetos, e a identificar quando utilizar o mesmo.

E um aprofundamento no *framework Flutter* com boas práticas e técnicas de desenvolvimento, onde é abordado estudos de casos de aplicações, para adquirir produtividade e assertividade ao desenvolver um sistema.

3.2.1 Ferramentas utilizadas para a pesquisa

Nessa seção será apresentado as ferramentas utilizadas nessa pesquisa:

- Computador utilizado: *Notebook Dell* latitude 5420;
- Celular utilizado: *Galaxy* m21s;
- *Android Studio Chipmunk* 2021.x para criação do *front-end*.
- *Intellij Idea* 221.x para criação do *back-end*.
- *Draw.io* para criação dos diagramas e ilustrações do sistema.
- Figma para criação de protótipos de telas do sistema.
- *SonarQube* para revisão e análise da qualidade de código.
- *Docker-Compose* para orquestração de contêineres do *sonarqube* e do *SGBD* onde estão armazenados seus dados.

3.3 Arquitetura de software do Projeto

3.3.1 MVVM x MVC x MVP

Para o desenvolvimento de arquitetura de software podem ser usadas várias abordagens conhecidas hoje no mercado, como o *MVVM*, *MVC*, *MVP* ou arquiteturas híbridas, que utiliza duas ao mesmo tempo.

Segundo Araújo (2021b), em seu livro, a arquitetura *MVVM* é utilizada para separar responsabilidades e facilitar a manutenção de um sistema, mantendo uma camada entre o modelo de negócios e a *View*[1]. Essa camada recebe requisições e as liga (*binding*) [2] aos seus responsáveis, impedindo a *View* de ter acesso direto à camada de negócios. Ainda há possibilidade de implementar *commands* [3] na camada de *ViewModel*, que reagem a interações do usuário. Sendo encontrada em aplicações *WPF* [4] e *Silverlight*, proprietárias da *Microsoft*.

Tal arquitetura tem como principal diferença o desacoplamento da *View* com o *ViewModel, onde várias Views podem receber dados de um mesmo ViewModel*, porém este não tendo informações das *Views*. Esse desacoplamento facilita o desenvolvimento de testes unitários para o sistema, pois não é necessário estar repetindo trechos de código em arquivos diferentes que possuem a mesma função.

Existe também o *MVC*, que é definido como um padrão que divide a aplicação em responsabilidades, onde o M se refere ao *Model* representando as regras de negócio, o V a *View*, se referindo a *interface* do usuário e o C o *Controller*, que realiza a comunicação entre as camadas (ARAÚJO, 2021b).

Tem como principal diferença todas as *Views* possuirem seu *Controller* próprio e uma divisão de camadas bem definidas.

E por fim, o *MVP* é uma derivação do *MVC*, melhorando a separação de interesses e sendo projetado para facilitar os testes unitários (ARAÚJO, 2021b). Tem como principal diferença possuir um *Presenter* para cada *View*, e a *View* delegando eventos ao mesmo, atualizando o estado de exibição, possuindo assim um certo acoplamento.

---

[1] É a visão do usuário, por onde o mesmo interage com o sistema.
[2] Termo utilizado para definir ligação dentro da arquitetura.
[3] comandos de interação do usuário.
[4] Subsistema gráfico no *.NET Framework* 3.0, que usa uma linguagem de marcação, conhecida como *XAML*.

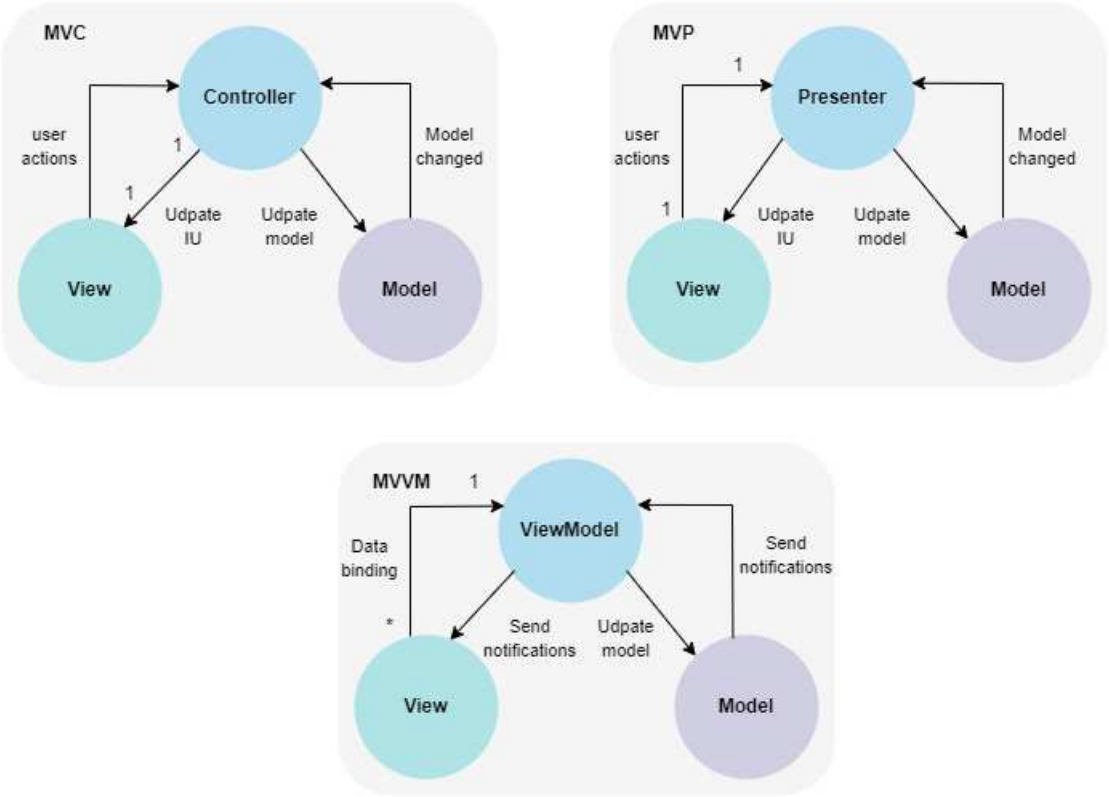

Figura 6 – Comparação entre as arquiteturas MVC, MVP e MVVM.

Fonte: Autoria Própria (2022).

Para esse trabalho foi escolhida a arquitetura *MVVM* no projeto *front-end* por causa da utilização de *Views* que possuem funcionalidades interligadas, como por exemplo o número de usuários em uma sala, onde manter um *ViewModel* para duas *Views* permite que esse número seja reutilizado de forma fácil, sem repetir trechos de códigos ou refazer testes.

3.3.2 Abordagem de design da Arquitetura

Conforme afirmado por Martin (2019), em seu livro, a arquitetura de um sistema é definida por limites especificados nele, pelas dependências que se aproximam desses limites, e pelos mecanismos físicos por quais os elementos são executados e se comunicam. A divisão de responsabilidades e de camadas tem um papel importante para o desacoplamento do sistema, possibilitando-o de suportar todos os casos de usos necessários, mesmo que não sabendo quais são eles, por exemplo através da aplicação de princípios como o *Princípio da Responsabilidade Única* e o *Princípio do Fechamento Comum*.

Sendo o Princípio da Responsabilidade Única definido como *"um módulo deve ter uma, e apenas uma, razão para mudar"* Martin (2019), de forma mais simples, significando apenas um arquivo-fonte, e o Princípio do Fechamento Comum como *"Reúna*

em componentes as classes que mudam pelas mesmas razões e nos mesmos momentos. Separe em componentes diferentes as classes que mudam em momentos diferentes e por diferentes razões." Martin (2019), significando que é preferível que modificações de classes ocorram somente em um componente em vez de serem distribuídas por vários componentes (MARTIN, 2019).

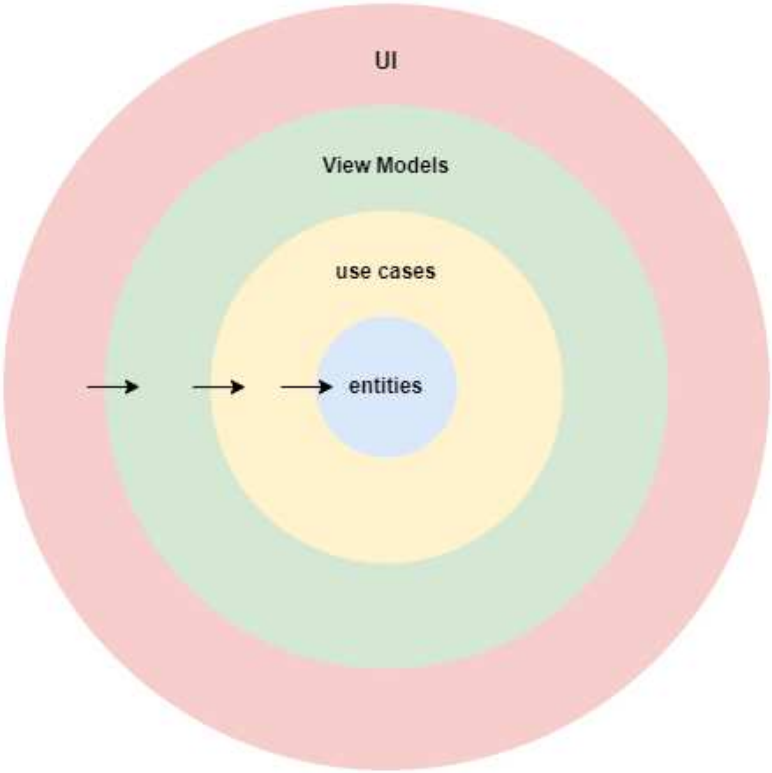

Figura 7 – Ilustração da arquitetura do sistema em camadas.

Fonte: Autoria Própria (2022).

Dessa forma, o sistema foi desenvolvido utilizando camadas horizontais e verticais desacopladas - (*UI, View Models, Use Cases* e *Entities*), onde a camada mais externa conhece a camada interna, mas a interna não conhece a externa, sendo possível visualiza-la em Figura 12.

3.4 Protocolo de Aplicação

O desenvolvimento e funcionamento do sistema é baseado em um protocolo de aplicação, que utiliza a estrutura do protocolo de comunicação para estabelecer a conexão e transportar as informações entre o *back-end* e o *front-end*, onde são interpretados. Possuindo tipos de mensagens para serem interpretadas, como mensagens de estado, entre os usuários e gerenciais.

As mensagens de estado são referentes as notificações do sistema, as mensagens entre os usuários ao conteúdo enviado entre os mesmos, e as mensagens gerenciais ao comportamento que deve ser efetuado tanto no *back-end* quanto no *front-end*, as ações quem podem ser efetuadas pelo remetente. Como por exemplo, quando um usuário envia uma mensagem em uma sala virtual pública, a mensagem gerencial indicará ao *back-end* a ação que ele deve executar, que é adicionar essa mensagem na sala destinatária, com o intuito de notificar todos os participantes que estão nesta sala no momento. Após o *back-end* processar essa mensagem gerencial, é enviada uma mensagem de estado para o *front-end* de todos os participantes, notificando que existe uma nova mensagem de usuário e atualizando a *view* com a mensagem do remetente.

Como é detalhado na UML de classes 11, as mensagens gerenciais e de estado possuem formatações diferentes para cada tipo de funcionalidade com a intenção de reduzir e otimizar o fluxo de dados trafegados entre o *back-end* e o *front-end*, permitindo o escalonamento do número de usuários dentro das salas e do sistema.

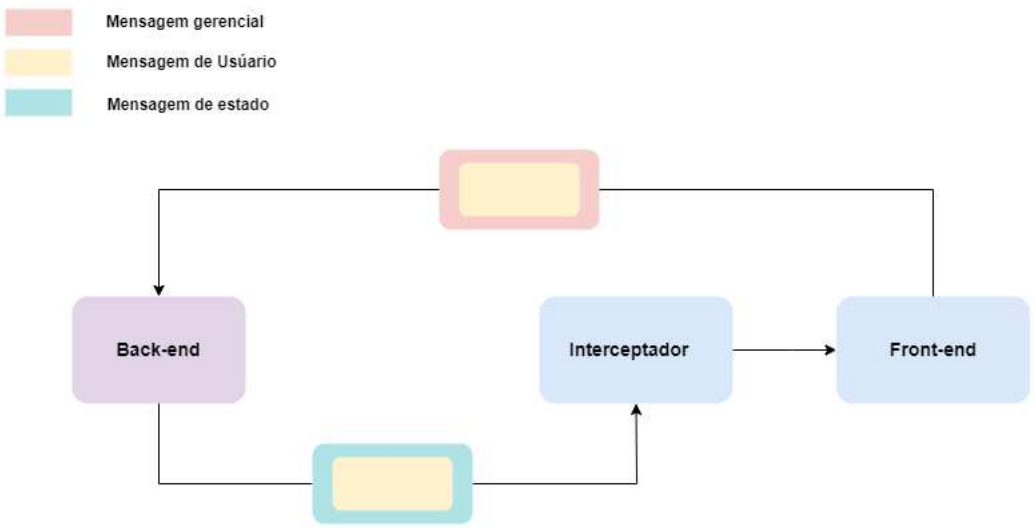

Figura 8 – Ilustração do protocolo de aplicação.

Fonte: Autoria Própria (2022).

3.5 Pseudocódigos do Sistema

Abaixo será descrito em pseudocódigos o funcionamento do protocolo de comunicação entre o *back-end* e o *front-end* desenvolvido para processamento de todas as possíveis ações do usuário. Sem esse protocolo haveriam informações trafegadas de forma desnecessária entre usuários remetentes e os destinatários, prejudicando o desempenho do serviço à medida que o número de usuários simultâneos fossem aumentando.

Algoritmo 1: Entrada na sala pública - Backend

input : Novo usúario de uma sala pública no Backend
output : Notificação para todos os usuários conectados no Frontend
data : instância do Socket
param : Parâmetro recebido do cliente

```
1  usuario ← Usuario(data.id, param.nickname, param.idade, param.genero,
     param.latitude, param.longitude);

2  for sala ∈ salasAtivas do
3      /*verifica se já existe um participante com esse nickname em alguma das
         salas já criadas.*/
       for participante ∈ sala.participantes do
4          if participante.nickname equal param.nickname then
5              /*atribuindo a variável booliana para usuário existente*/
               salaExiste ← true
6          end
7      end
8      if !salaExiste then
9          /*atribuindo a variável booliana para usuário existente*/
           for sala ∈ salas do
10             for salaAtiva ∈ salasAtivas do
11                 if sala.idSala equal salaAtiva.idSala then
12                     /*atribuindo a variável booliana para usuário existente*/
                       salaExiste ← true
                       salaAtiva.participantes.adicionar(usuario);
13                 end
14             end
15             if !salaExiste and sala.idSala equal param.idSala then
16                 salaAtiva.participantes.adicionar(usuario);
                   salasAtivas.adicionar(sala);
17             end
18             salaExiste ← true
19         end
20     end
21 end
22 dadoInitialusuario ← dadoSalaPublica(param.idSala, param.nickname,
     usuario, param.nomeSala, tipoEventoSocket);
   data.entrar(param.idSala);
   data.emitir('entrar_sala_publica', dadoInitialusuario);
```

Para o desenvolvimento da arquitetura foi utilizado como referência o livro Comer (2016), sendo base para a construção de *endpoints*, tipos de retorno, argumentos e nomes de funções.

No Algoritmo 1 é descrito como é o comportamento da entrada de um usuário em uma sala pública no *back-end*. Onde, ao receber a requisição, será feita a instância de um usuário, passado os dados que foram recebidos por parâmetro e verificado se já existe nessa sala o *nickname* informado. Caso não exista, será feita uma busca por essa sala informada na lista de salas ativas e inserido o usuário na mesma, caso não exista e essa sala não estiver em salas ativas, é adicionado o usuário na sala e a mesma é adicionada à lista de salas ativas, e caso exista, o usuário não será inserido na sala. No *front-end* existe uma validação que informará o usuário que esse *nickname* já está em uso nessa sala e exigirá que seja modificado e tente entrar novamente. Por fim, se o usuário entrar, o mesmo terá o id do soquete adicionado a sala e todos os usuários conectados serão notificados que o mesmo entrou na sala.

Algoritmo 2: Troca de mensagem em sala pública - Backend

input : Mensagem enviada para uma sala pública no Backend
output : Notificação para os usuários conectados na sala de destino no Frontend
data : instância do Socket
param : Parâmetro recebido do cliente

```
1 dadoEnvioMensagem ← DadoEnvioMensagem(data.idSala, geradorId(),
    param.criadoEm, param.nomeSala, param.mensagemTexto, param.usuario,
    tipoEventoSocket);

2 for sala ∈ salasAtivas do
3 |   /*busca a sala na qual foi enviada a mensagem dentro das salas ativas*/
  |    if sala.idSala equal param.idSala then
4 |   |   /*adiciona a mensagem na lista de mensagens da sala*/
  |   |    sala.mensagens.adicionar(dadoEnvioMensagem);
5 |   end
6 end
7 data.broadcast.para(param.idSala).
    emitir('mensagem_publica', dadoEnvioMensagem);
```

No Algoritmo 2 é descrito como é o comportamento de troca de mensagem em uma sala pública no *back-end*. Onde, ao receber a requisição, será feita à instância do modelo de mensagens enviadas e passado os dados que foram recebidos por parâmetro. Será feita uma busca por essa sala informada na lista de salas ativas e a mensagem enviada será inserida na lista de mensagens que existe dentro da instância da sala. Por fim, todos os usuários conectados a sala informada receberão a mensagem.

Algoritmo 3: Saída da sala pública - Backend

input : Usúario solicita saida de sala pública no Backend
output : Notificação para todos os usuários conectados no Frontend
data : instância do Socket
param : Parâmetro recebido do cliente

```
1  usuario ← Usuario(data.id, param.nickname, param.idade, param.genero,
     param.latitude, param.longitude);

2  for sala ∈ salasAtivas do
3      /*verifica se já existe um participante com esse nickname em alguma das
         salas já criadas.*/

4      for sala ∈ salasAtivas do
5          /*busca a sala que o usuário pertence.*/
           if sala.idSala equal param.idSala then
6              /*remove-o da lista de usuários da sala*/ for
                 participante ∈ sala.participantes do
7                  if participante.idUsuario equal param.idUsuario then
8                      sala.participantes.remover(participante);
9                  end
10             end
11         end
12     end
13     dadoSaidaSala ← dadoSalaPublica(param.idSala, param.nickname,
         usuario, param.nomeSala, tipoEventoSocket);
       data.sair(param.idSala);
       data.emitir('sair_sala_publica', dadoSaidaSala);
14 end
```

No Algoritmo 3 é descrito como é o comportamento da saída de um usuário em uma sala pública no *back-end*. Onde, ao receber a requisição, é feita a instância de um usuário, passado os dados que foram recebidos por parâmetro. Será feita uma busca por essa sala informada na lista de salas ativas e removido o usuário da mesma. Por fim, o mesmo terá o id do soquete removido da sala e todos os usuários conectados serão notificados que o mesmo saiu da sala.

Algoritmo 4: Interceptador de dados - Frontend

input : resposta do Backend
output : Interpretação da resposta e mudança de estado do aplicativo
param : Parâmetro recebido do backend

```
1  switch param.tipoEventoSocket do
2      case tipoEventoSocket.atualizar_salas do
3          /*retorna o estado de salas carregadas com sucesso, carregando a lista
            de salas para o usuário*/
           break;
4      case tipoEventoSocket.entrar_sala_publica do
5          /*retorna o estado de entrada em sala pública, notificando a
            sala onde o usuário está* break;
6      case tipoEventoSocket.mensagem_publica do
7          /*retorna o estado de mensagem pública, adicionando a mesma a
            sala onde o usuário está*/
           break;
8      case tipoEventoSocket.saida_sala_publica do
9          /*retorna o estado de saída de pública, notificando a sala onde
            o usuário está*/
           break;
10     otherwise do
11         break;
12     end
13 end
```

No Algoritmo 4 é descrito como é o comportamento do interceptador de mensagens no *front-end*. Onde, ao receber as respostas do *back-end*, será interpretado qual o tipo de evento foi recebido, e com base no mesmo é acionado um *case* do *switch*, que disparará um novo estado no sistema que seguirá um fluxo. Esse algoritmo é um dos mais importantes para a implementação do sistema, pois através dele é possível trafegar estruturas de dados para cada evento, possibilitando a redução de pacotes e otimização do protocolo.

4 Resultados

Nesse capítulo é apresentado os resultados alcançados no desenvolvimento do trabalho. Segue abaixo os resultados obtidos a partir da elicitação de requisitos, do escopo do projeto, diagramas e ilustrações do sistema.

4.1 Arquitetura Ilustrativa do Sistema

A figura abaixo representa uma ilustração de como o sistema foi projetado para funcionar, onde as requisições feitas no *front-end* serão processadas por um *back-end* que gerencia as salas criadas por geolocalização, as mensagens trocadas e os usuários das mesmas.

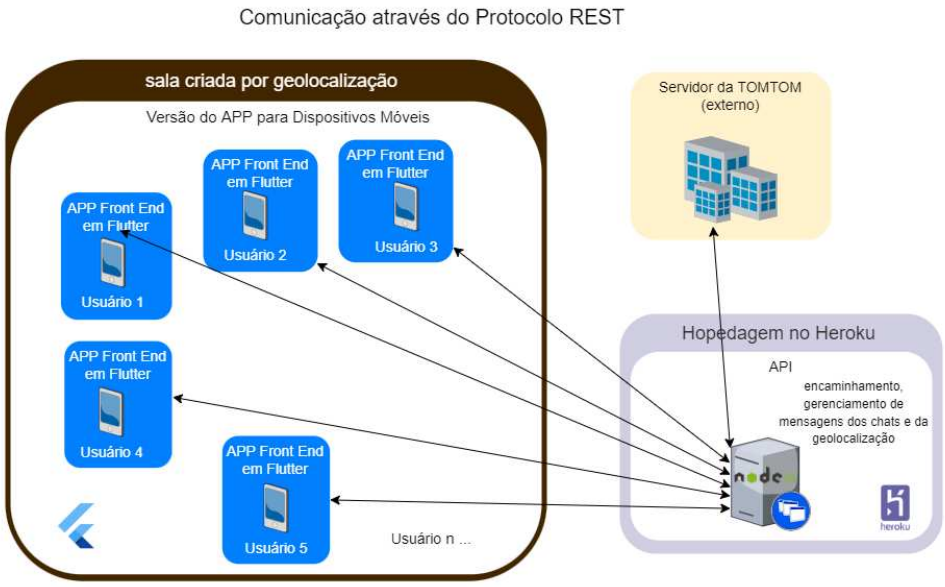

Figura 9 – Arquitetura Ilustrativa do Aplicativo.

Fonte: Autoria Própria (2021).

4.2 *Unified Model Language - Linguagem de Modelagem Unificada (UML)*

Guedes (2018) define, em seu livro, *UML* é definida como uma linguagem visual para modelagem de *softwares*, sendo utilizada na engenharia de *software* para definir características de sistemas orientados à objetos para representar suas necessidades físicas e lógicas como requisitos, comportamentos, estruturas lógicas, entre outras, através de diagramas e modelos visuais (GUEDES, 2018).

4.2.1 UML de Caso de Uso

O diagrama de caso de uso apresenta as funcionalidades do sistema, seu comportamento e como os usuários devem interagir com o mesmo (GUEDES, 2018).

Abaixo segue o modelo *UML* de caso de uso do sistema, retratando todos os comportamentos do mesmo e quais são as exigências necessárias para executar tais procedimentos, funções essas que são complementadas e descritas no documento de requisitos funcionais.

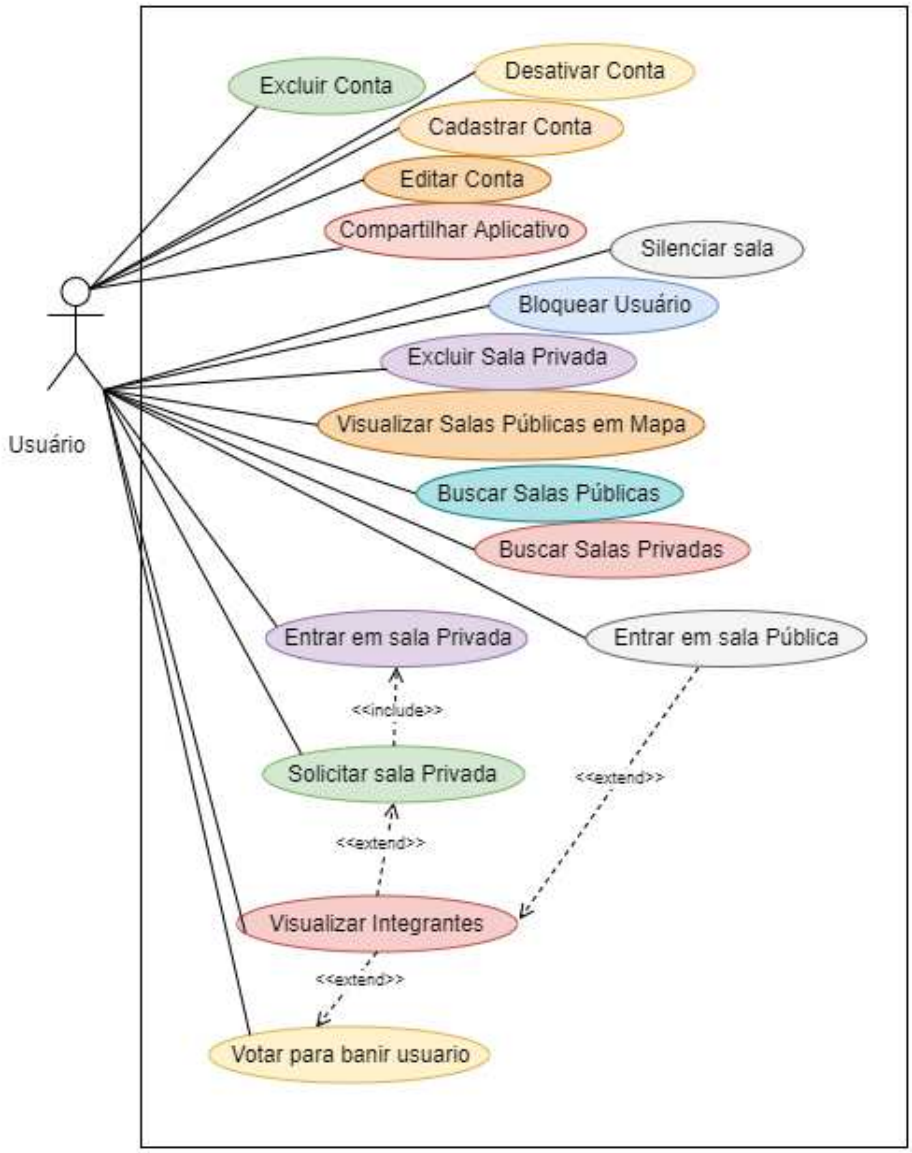

Figura 10 – UML de Caso de Uso do Sistema.

Fonte: Autoria Própria (2022).

4.2.2 UML de Classes

O diagrama de classes permite a visualização das classes do sistema, seus atributos e métodos. Demonstra o fluxo de informações entre as classes, a partir de uma abordagem orientada a objetos (GUEDES, 2018).

Abaixo segue o diagrama de classes do sistema.

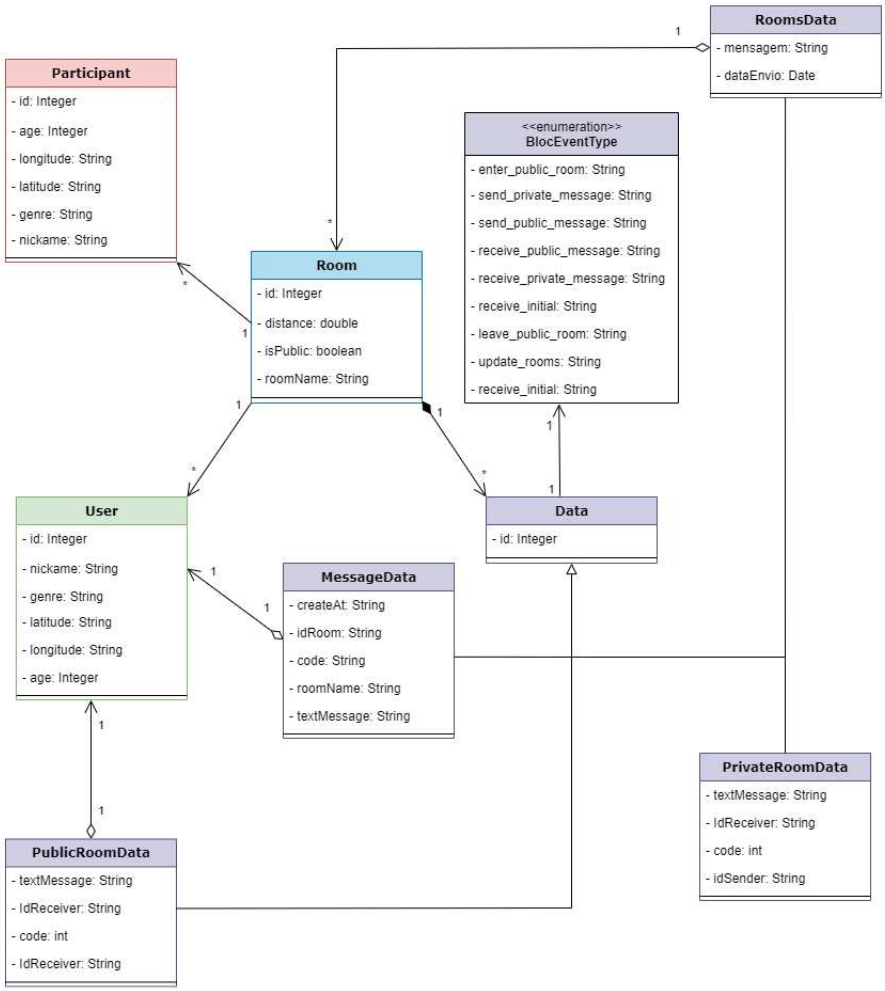

Figura 11 – UML de Classe do Sistema.

Fonte: Autoria Própria (2022).

4.3 UML de Pacotes

O diagrama de pacotes representa como os componentes do sistema estão divididos dentro do projeto, suas estruturas, subsistemas, características e relacionamentos (GUEDES, 2018).

Segue abaixo o diagrama de pacotes do sistema.

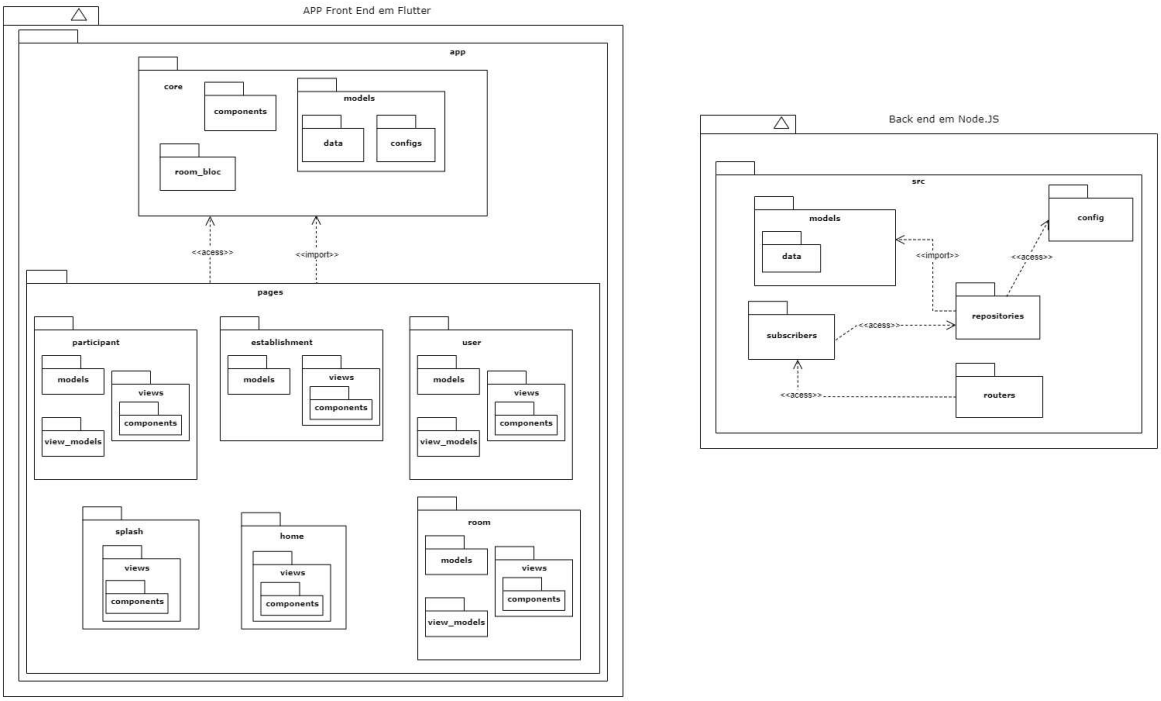

Figura 12 – UML de Pacotes do Sistema.

Fonte: Autoria Própria (2022).

4.4 Protótipos de Telas Desenvolvidas

Figma é uma ferramenta web de design de interface, sendo escolhida pela sua versatilidade, funções de compartilhamento de tela em tempo real, ferramentas e bibliotecas, contendo *templates* e componentes prontos para serem anexados a prototipação (INTERATIVA, 2019).

4.4.1 Telas Iniciais do Sistema

Abaixo temos as telas para cadastro no sistema, onde os usuários clientes do estabelecimento e representantes dos mesmos podem efetuar o cadastro preenchendo os dados solicitados.

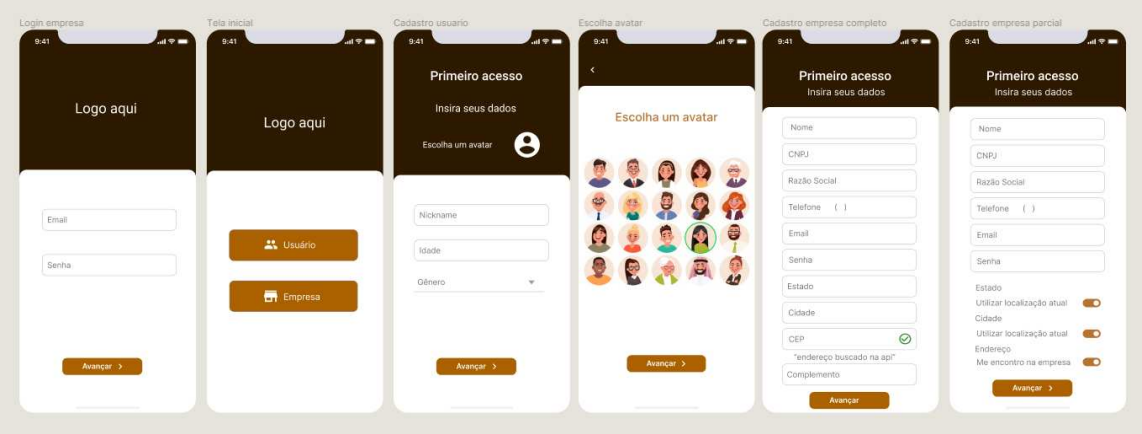

Figura 13 – Telas Iniciais do Sistema.

Fonte: Autoria Própria (2021).

4.4.2 Telas de salas Públicas e Privadas

Abaixo temos as telas de navegação entre as salas no sistema, onde os usuários podem visualizar as salas públicas em formato de lista disponíveis num raio de 5km, e ao lado as salas privadas já iniciadas, tendo as opções de gerenciamento das mesmas.

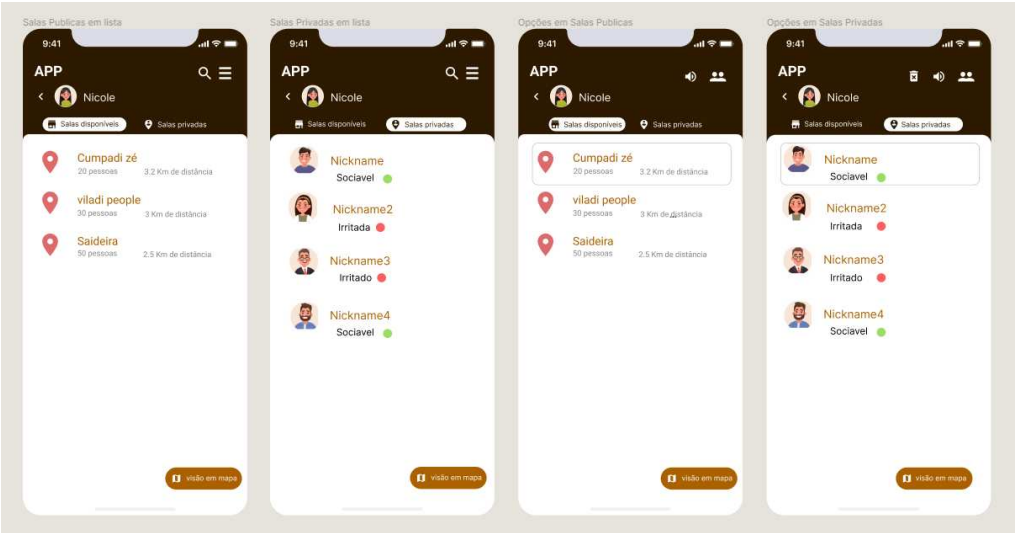

Figura 14 – Telas de Salas Públicas e Privadas.

Fonte: Autoria Própria (2021).

4.4.3 Telas de Salas Publicas em Mapa e Chat Público

Abaixo temos a tela de salas públicas em mapa do sistema disponíveis num raio de 5km, e ao lado temos o chat da mesma.

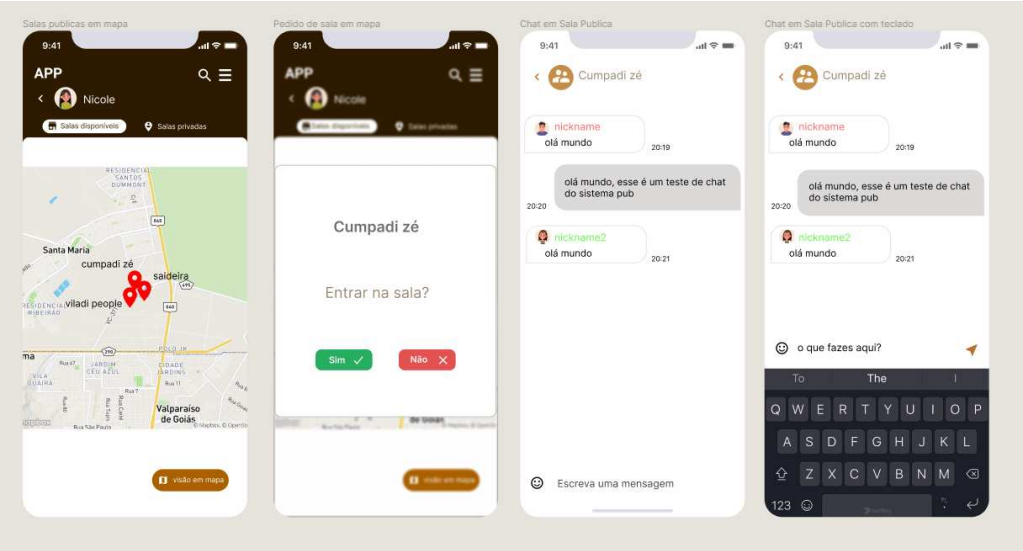

Figura 15 – Telas de Salas Publicas em Mapa e Chat Público.

Fonte: Autoria Própria (2021).

4.4.4 Telas de Integrantes da Sala Pública e Chat Privado

Abaixo temos a tela de integrantes da sala pública, onde é possível selecionar um usuário específico e solicitar uma sala privada, caso o mesmo aceite, e ao lado temos a tela de troca de mensagens da sala privada.

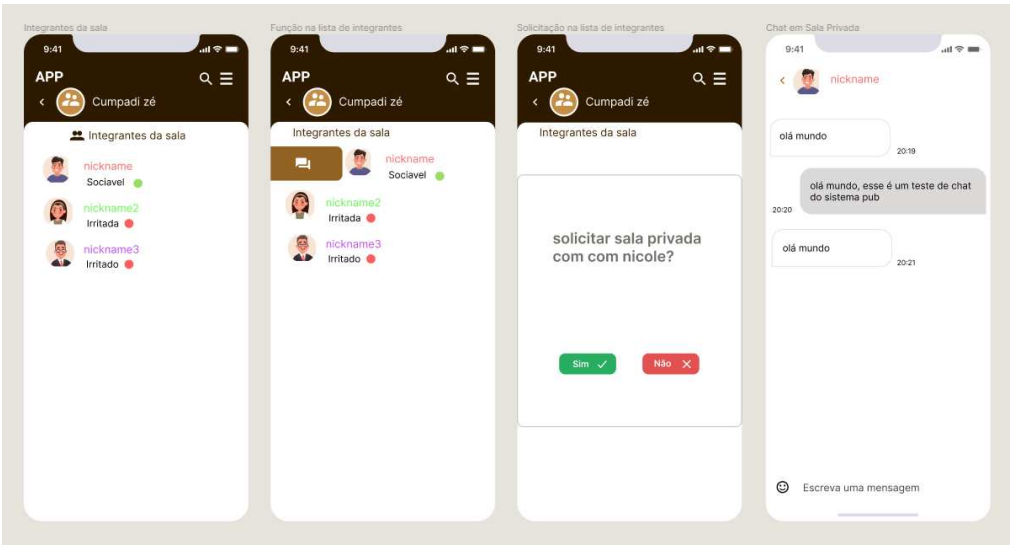

Figura 16 – Telas de Integrantes da Sala Pública e Chat Privado.

Fonte: Autoria Própria (2021).

4.5 Teste de Qualidade de Código do Front-end e Back-end

Para realizar esse teste foi utilizado o *SonarQube* em um contêiner *docker* gerenciado pela ferramenta *Docker Compose*, utilizando o *PostgreSQL* para armazenar os dados gerenciados pelo *SonarQube*.

Contextualizando, o *Docker* é uma plataforma que permite virtualização de *kernels*[1] em contêineres, contendo somente o necessário pra executar determinado serviço e isolando os mesmos de qualquer interferência entre eles ou com o sistema operacional onde é feita a virtualização, diferente da máquina virtual que possui um sistema operacional completo sendo virtualizado, consumindo mais recursos.

Mella (2020) Define *Docker Compose* como "O *Docker Compose* é uma ferramenta de administração de contêineres", sendo utilizado para administrar um contêiner com o *PostgreSQL*, outro com o *SonarQube*, portas de acesso, dependências e volumes.

Segue abaixo uma comparação ilustrativa entre virtualização de contêineres e uma máquina virtual contendo esses mesmos serviços.

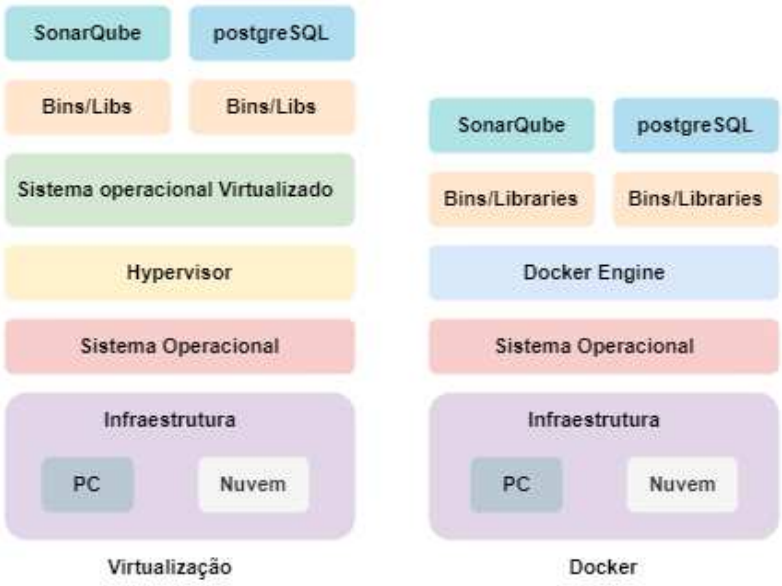

Figura 17 – Máquina virtual e contêineres utilizando os mesmos serviços.

Fonte: Autoria Própria (2022).

Segundo sua Documentação Oficial, "O *SonarQube* é uma ferramenta automática de revisão de código para detectar *bugs*[2], vulnerabilidades e *code smells*[3] em seu código" Sonarqube,org (2022), fazendo a análise desses aspéctos e atribuindo uma nota, como por exemplo na segurança é atribuído uma nota de "A" a "E".

[1] Núcleo central de um sistema operacional.
[2] Bug é utilizado para se referir à uma falha no sistema.
[3] Code smell é qualquer característica no código que indique uma possível má qualidade de escrita.

Abaixo segue os resultados da análise realizada pelo *SonarQube* nos projetos *front-end* e *back-end* do sistema.

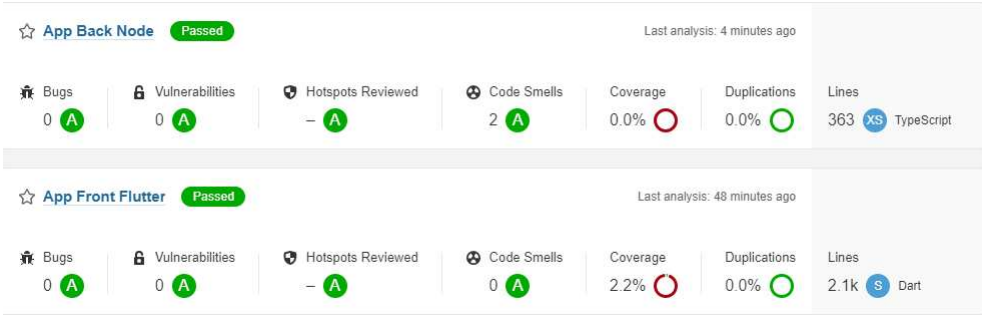

Figura 18 – Resultado dos testes realizados no SonarQube.

Fonte: Autoria Própria (2022).

Não sendo encontrado nenhum *bug*, vulnerabilidade de segurança ou duplicidade de código em ambos os sistemas. No *back-end* foram encontrados 2 *Code Smells*, porém sendo atribuído uma nota "A", estando dentro do aceitável para o desenvolvimento de um sistema. Não há nenhum *Hotspots Reviewed* (Pontos de Acesso revisados) nos projetos, sendo atribuído a nota padrão "A", e como não foram realizados testes unitários de cobertura de código, esse atributo não foi avaliado. Ambos os projetos receberam nota "A" de qualidade de código, sendo aprovados pelo *SonarQube*.

5 Conclusão

Este trabalho tem como objetivo desenvolver uma nova abordagem de sistema de comunicação em salas virtuais em ambientes públicos baseados na geolocalização de usuários, apresentando uma pesquisa bibliográfica e descrevendo o processo metodológico, documental, de desenvolvimento e aplicação.

Durante o processo de desenvolvimento foram enfrentados alguns obstáculos de integração do *back-end* com o *front-end* devido a versão da biblioteca utilizada no *back-end* para criação da comunicação não ser compatível com a versão utilizada no *front-end*, e da escolha de abordagem de gerenciamento de estado no *framework Flutter* não ter satisfeito a implementação do *socket* no lado do cliente, sendo necessário mudar a abordagem, o que acarretou em uma perda demasiada de tempo.

Houve também uma dificuldade em manter o funcionamento das salas virtuais estáveis e consistentes com o aumento de usuários no período inicial do sistema, sendo resolvido com o desenvolvimento e implementação do protocolo de aplicação.

Ao fim do desenvolvimento do aplicativo, foi realizado um estudo de caso onde o mesmo foi publicado e disponibilizado na Google Play 36 para ser utilizado na UNITINS Campus Palmas com o intuito de recolher opiniões de usuários anônimos para validação da viabilidade da implementação do sistema com o intenção de falicitar a comunicação interpessoal. Obtendo respostas de melhorar a aplicação do sistema, possíveis concorrentes e situações de utilização.

Conclui-se que a proposta do sistema é viável para favorecer a interação de pessoas e facilitar a comunicação em ambientes públicos.

5.1 Trabalhos Futuros

Sugestão de próximos passos que podem ser realizados em trabalhos futuros:

- Correção de *bugs* do sistema e criação de testes unitários;
- Certificar que o sistema está em conformidade com as normas da LGPD.

Referências

AFFDE. Estatísticas do usuário do whatsapp 2021: quantas pessoas usam o whatsapp? [Online; accessed 22-novembro-2021]. 2021. Disponível em: <https://www.affde.com/pt/whatsapp-users.html>.

ANDRADE, A. P. de. O que é flutter? [Online; accessed 23-novembro-2021]. 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-flutter>.

ANDRADE, A. P. de. O que é o fuchsia? [Online; accessed 23-novembro-2021]. 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-fuchsia>.

ARAÚJO, E. C. de. *Aprofundando em Flutter - Desenvolva aplicações Dart com Widgets*. [S.l.]: CASA DO CÓDIGO, 2021. ISBN 978-65-86110-75-3.

ARAÚJO, E. C. de. *Xamarin Forms e MVVM: Persistência local com Entity Framework Core*. [S.l.]: CASA DO CÓDIGO, 2021. ISBN 978-85-94188-98-4.

BASEFLOW.COM. Biblioteca geolocator para flutter. [Online; accessed 03-junho-2022]. 2018. Disponível em: <https://pub.dev/packages/geolocator>.

BLOCLIBRARY.DEV. Documentação oficial do bloc. [Online; accessed 03-junho-2022]. 2018. Disponível em: <https://bloclibrary.dev>.

BUNGART, J. W. *Redes de computadores: Fundamentos e protocolos*. [S.l.]: Editora SESI-Serviço Social da Indústria, 2017. ISBN 978-85-83937-64-7.

CAMPOS, G. F. B. de. Sistema móvel na plataforma phonegap para compartilhamento de geolocalização integrado a rede social. *Universidade Regional de Blumenau*, 2015.

CAVALCANTE, P. H. A. Introdução a typescript: o que é e como começar? [Online; accessed 24-novembro-2021]. 2021. Disponível em: <https://blog.geekhunter.com.br/introducao-a-typescript/>.

COMER, D. E. *Redes de Computadores e Internet*. [S.l.]: Bookman Editora, 2016. ISBN 978-85-60031-36-8.

CURY, A. J. *Ansiedade: como enfrentar o mal do século*. [S.l.]: Saraiva Educação SA, 2017.

FERREIA, J. Entenda as diferenças entre os protocolos tcp e udp. [Online; accessed 22-novembro-2021]. 2021. Disponível em: <https://www.eletronet.com/entenda-as-diferencas-entre-os-protocolos-tcp-e-udp/>.

FLUTTER.DEV. Biblioteca url launcher para flutter. [Online; accessed 03-junho-2022]. 2019. Disponível em: <https://pub.dev/packages/url_launcher>.

FLUTTER.DEV. Shared preferences plugin. [Online; accessed 03-junho-2022]. 2022. Disponível em: <https://pub.dev/packages/shared_preferences>.

FONSECA, J. J. S. da. *Apostila de metodologia da pesquisa científica*. [S.l.]: João José Saraiva da Fonseca, 2002.

GONÇALVES, A. K.; FRANCISCO, G. de S. Aplicativo para a geolocalização de ônibus e transporte de estudantes. *Universidade Unisul*, 2020.

GONÇALVES, T. As maiores redes sociais em 2021. [Online; accessed 24-novembro-2021]. 2021. Disponível em: <https://etus.com.br/blog/as-maiores-redes-sociais-em-2021/>.

GROUP, M. About tinder. [Online; accessed 22-novembro-2021]. 2021. Disponível em: <https://tinder.com/pt/about-tinder>.

GUEDES, G. T. *UML 2-Uma abordagem prática*. [S.l.]: Novatec Editora, 2018. ISBN 978-85-75226-46-9.

HOLANDA, D. S. F. Internet. [Online; accessed 24-novembro-2021]. 2016. Disponível em: <https://medium.com/@aturc.jornalismo/o-que-%C3%A9-a-internet-ce7ee2be78f0>.

INTERATIVA, S. Figma: uma nova ferramenta para design de interface que está ganhando o mercado | sirius interativa. [Online; accessed 14-novembro-2021]. 2019. Disponível em: <https://medium.com/@Sirius_/figma-uma-nova-ferramenta-para-design-de-interface-que-est%C3%A1-ganhando-o-mercado-sirius-interativa-2e78e0905b44>.

KNOTH, P. Telegram tem recorde de novos usuários e garante: "não vamos falhar com você. [Online; accessed 22-novembro-2021]. 2021. Disponível em: <https://tecnoblog.net/502902/telegram-tem-recorde-de-novos-usuarios-e-garante-nao-vamos-falhar-com-voce/>.

KUROSE, J. F. et al. *Redes de Computadores ea Internet: uma abordagem top-down*. [S.l.]: Pearson Addison Wesley, 2010. ISBN 978-85-81436-77-7.

LOUREIRO, R. Pesquisa revela os aplicativos de mensagens mais utilizados no brasil. [Online; accessed 24-novembro-2021]. 2019. Disponível em: <https://exame.com/tecnologia/pesquisa-revela-os-aplicativos-de-mensagens-mais-utilizados-no-brasil/>.

MARINHO, L. H. *Iniciando com Flutter Framework - Desenvolva aplicações móveis no Dart Side!* [S.l.]: CASA DO CÓDIGO, 2020. ISBN 978-65-86110-26-5.

MARTIN, R. C. *Arquitetura Limpa: O guia do artesão para estrutura e design de software*. [S.l.]: Alta Books Editora, 2019. ISBN 978-85-50816-00-5.

MELLA, L. Docker e docker compose um guia para iniciantes. [Online; accessed 11-junho-2022]. 2020. Disponível em: <https://dev.to/ingresse/docker-e-docker-compose-um-guia-para-iniciantes-48k8>.

MOREIRA, L. A. L. Beaver: um aplicativo web para localizar prestadores de serviço por geolocalização. *Universidade Regional de Blumenau*, 2017.

OFICIAL, D. P. Dart overview - Documentação Oficial | Dart Pub Oficial. [Online; accessed 14-novembro-2021]. 2011. Disponível em: <https://dart.dev/overview>.

OFICIAL, F. FAQ Flutter - Documentação Oficial | Flutter Oficial. [Online; accessed 14-novembro-2021]. 2017. Disponível em: <https://docs.flutter.dev/resources/faq>.

PÁDUA, W. d. *Engenharia de software: fundamentos, métodos e padrões*. [S.l.: s.n.], 2003. ISBN 978-85-21616-50-4.

REIS, F. dos. Curso de redes – protocolo tcp (transmission control protocol). [Online; accessed 23-novembro-2021]. 2015. Disponível em: <http://www.bosontreinamentos.com.br/redes-computadores/curso-de-redes-protocolo-tcp-transmission-control-protocol/>.

REIS, F. dos. Curso de redes – protocolo udp (user datagram protocol). [Online; accessed 23-novembro-2021]. 2016. Disponível em: <http://www.bosontreinamentos.com.br/redes-computadores/curso-de-redes-protocolo-udp-user-datagram-protocol/>.

SOCKET.IO. Documentação do socket.io| what socket.io is. [Online; accessed 21-novembro-2021]. 2014. Disponível em: <https://socket.io/docs/v4/>.

SONARQUBE.ORG. Sonarqube documentation. [Online; accessed 11-junho-2022]. 2022. Disponível em: <https://docs.sonarqube.org/latest/>.

TANENBAUM, A. S. Redes de computadores. 5ª. *Edição. Editora Campus*, 2003.

TEDESCO, K. Uma introdução a tcp, udp e sockets. [Online; accessed 23-novembro-2021]. 2019. Disponível em: <https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets>.

WHATSAPP. Features of whatsapp. [Online; accessed 22-novembro-2021]. 2021. Disponível em: <https://www.whatsapp.com/features>.

6 Anexos

6.1 Anexo A - Questionário de Validação de Sistema de Comunicação Baseado em Salas Virtuais

6.1.1 Esclarecimento Sobre o Questionário

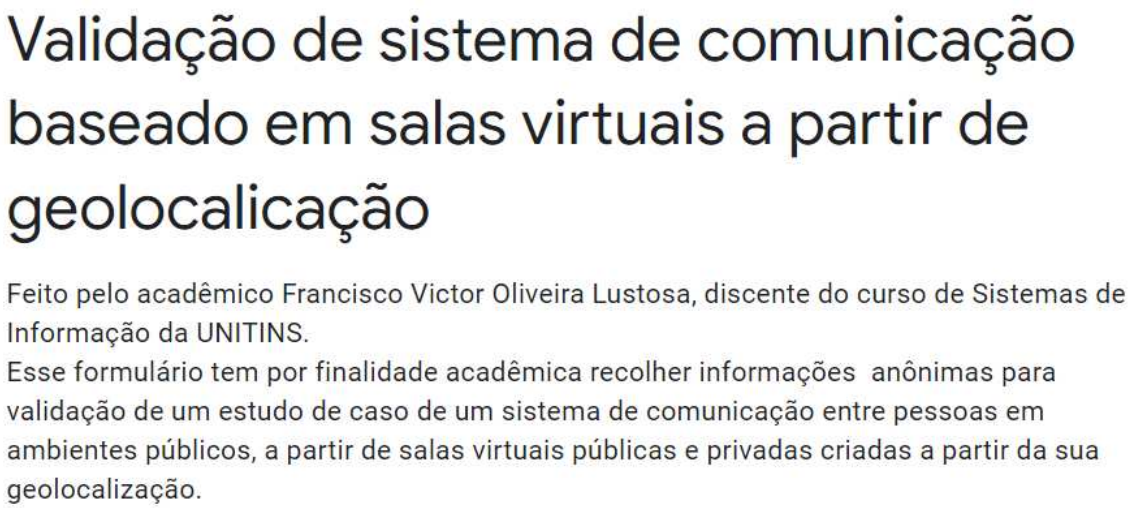

Figura 19 – Cabeçario do Questionário

Fonte: Google Forms.

6.1.2 Perguntas do Questionário

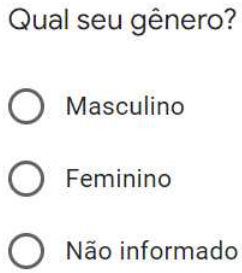

Figura 20 – Pergunta 1

Fonte: Google Forms.

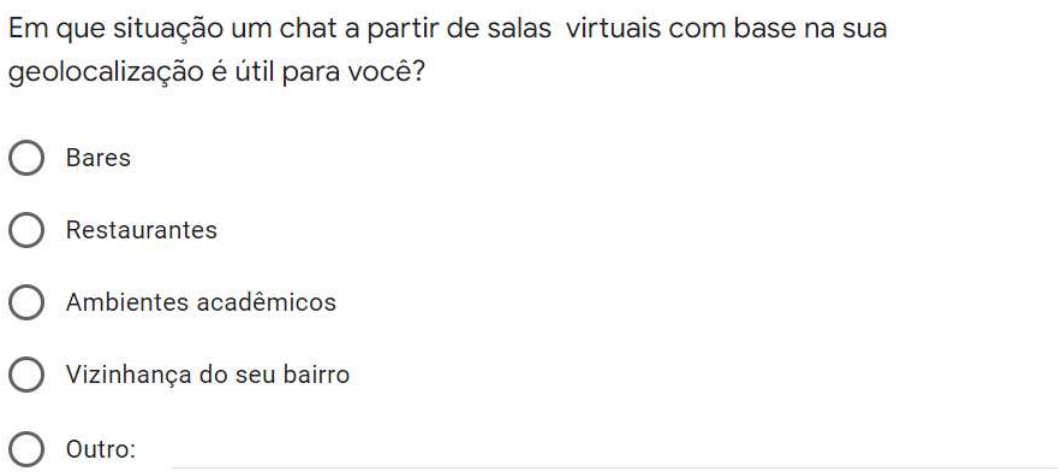

Figura 21 – Pergunta 2

Fonte: Google Forms.

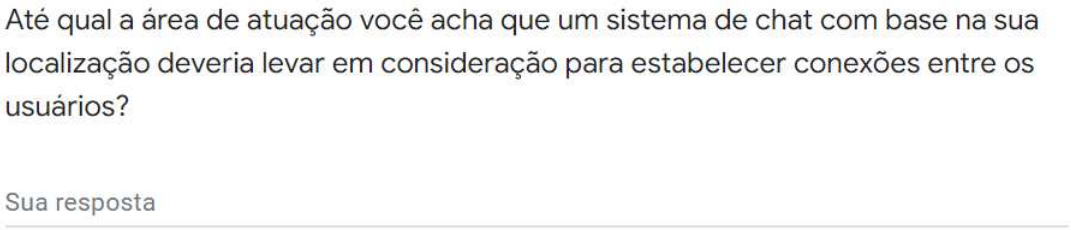

Figura 22 – Pergunta 3

Fonte: Google Forms.

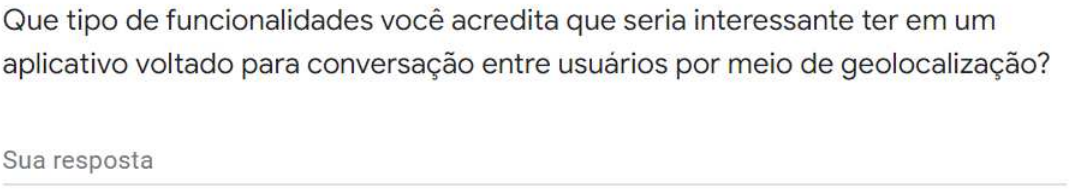

Figura 23 – Pergunta 4

Fonte: Google Forms.

Que informações para você seriam pertinentes para identificar usuários dentro de uma sala virtual?

- ◯ Hobbies
- ◯ Gostos Pessoais
- ◯ Profissão
- ◯ Outro: _____

Figura 24 – Pergunta 5

Fonte: Google Forms.

E que situações seriam interessantes um filtro de estabelecimentos por áreas de interesses?

Sua resposta

Figura 25 – Pergunta 6

Fonte: Google Forms.

Quais seriam os principais concorrentes que você imagina para esse sistema?

Sua resposta

Figura 26 – Pergunta 7

Fonte: Google Forms.

Você pagaria pela funcionalidade de conversar de forma privada com pessoas desconhecidas que fazem parte de uma mesma sala virtual geolocalizada?

- ◯ Sim
- ◯ Não
- ◯ Outro: _____

Figura 27 – Pergunta 8

Fonte: Google Forms.

6.1.3 Respostas do Questionário

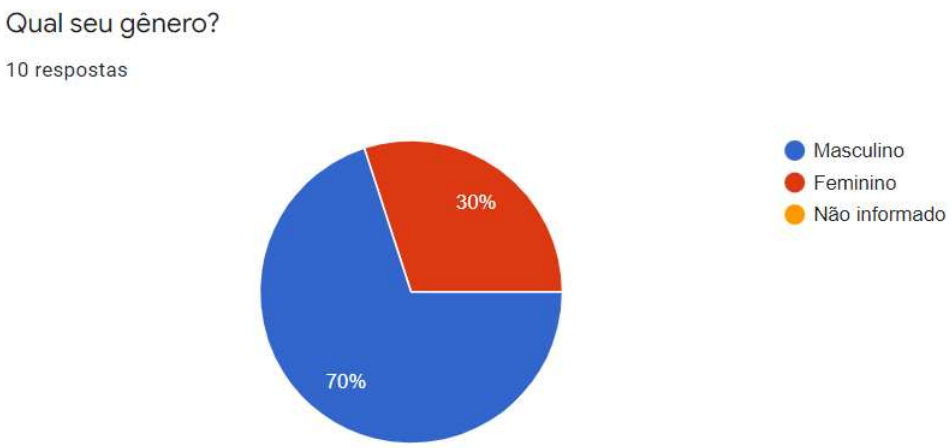

Figura 28 – Resposta 1

Fonte: Google Forms.

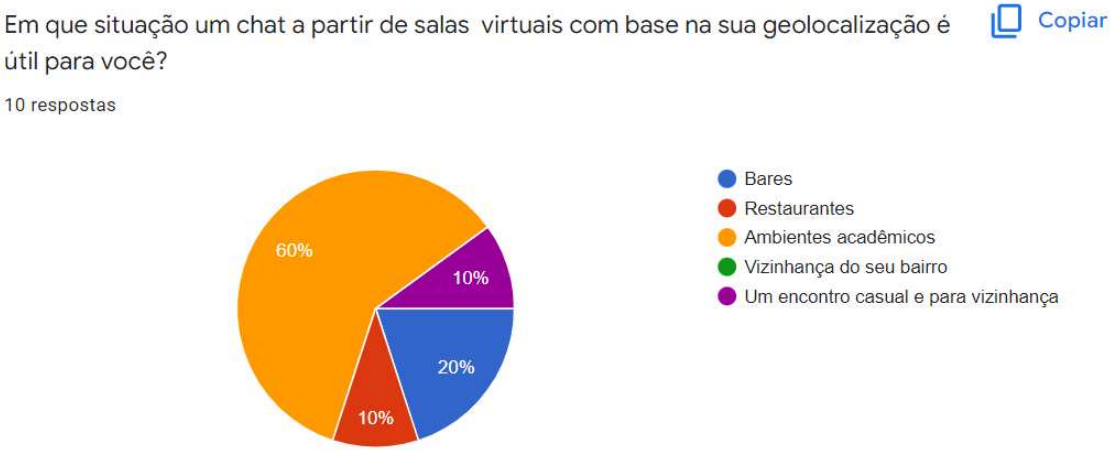

Figura 29 – Resposta 2

Fonte: Google Forms.

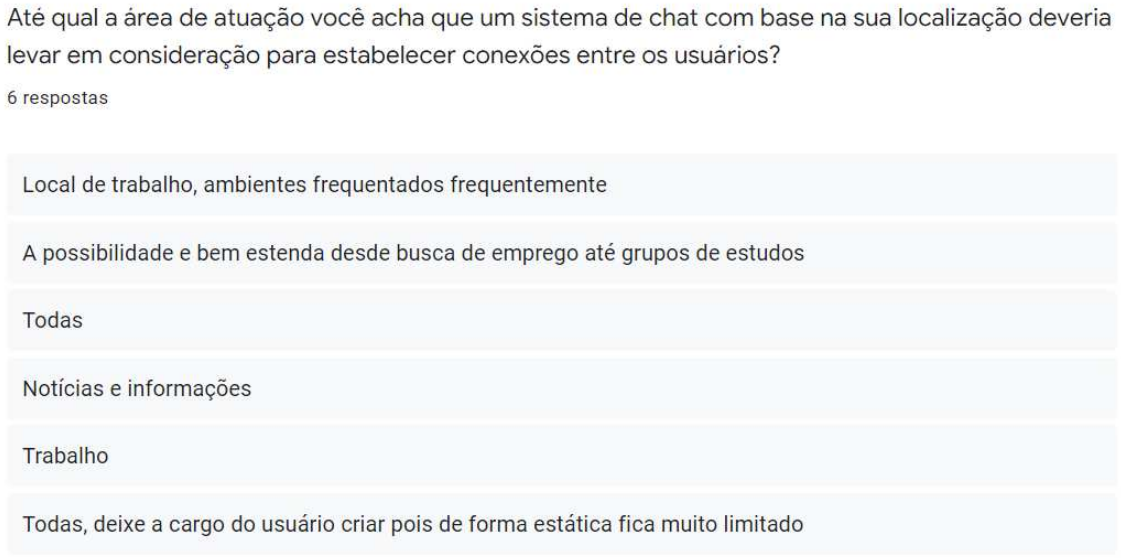

Figura 30 – Resposta 3

Fonte: Google Forms.

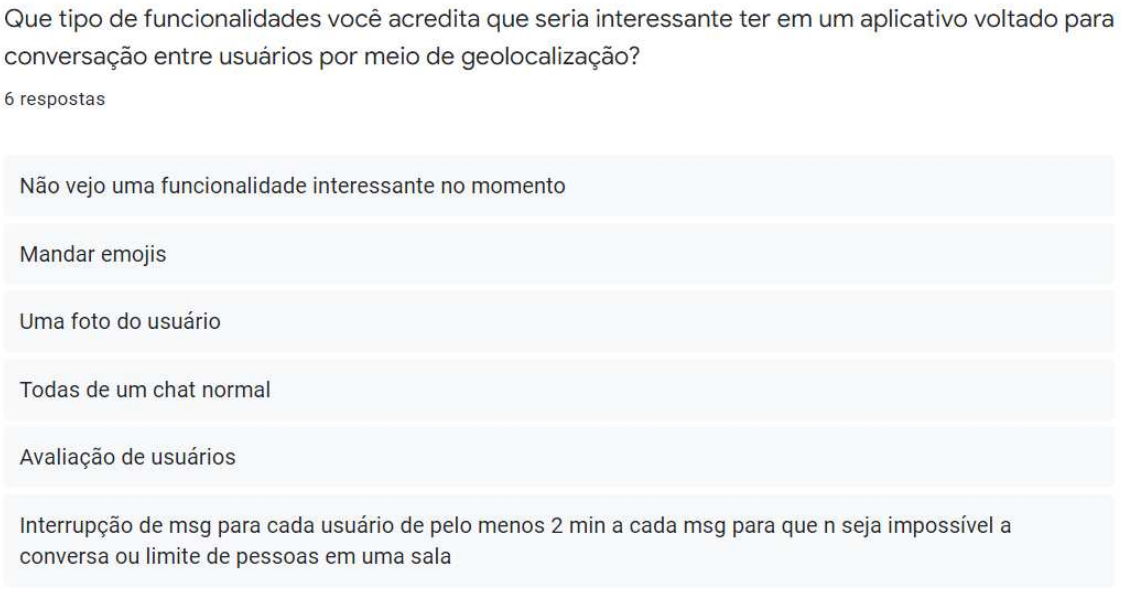

Figura 31 – Resposta 4

Fonte: Google Forms.

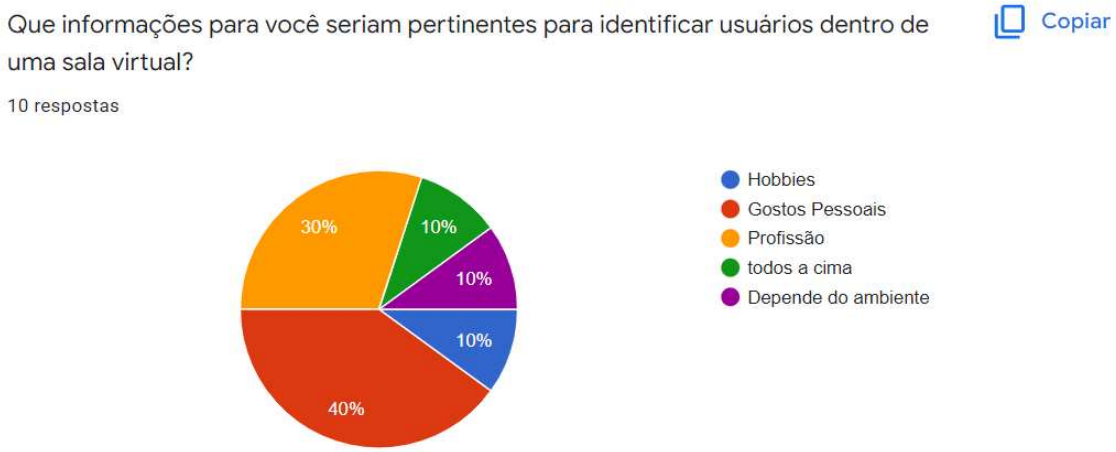

Figura 32 – Resposta 5

Fonte: Google Forms.

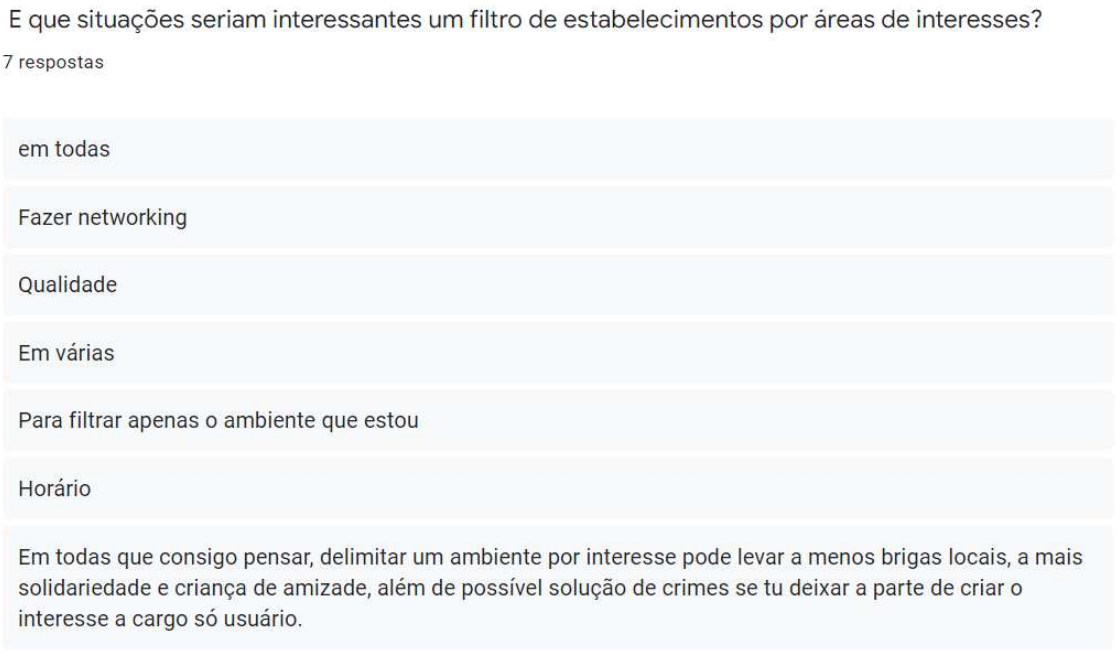

Figura 33 – Resposta 6

Fonte: Google Forms.

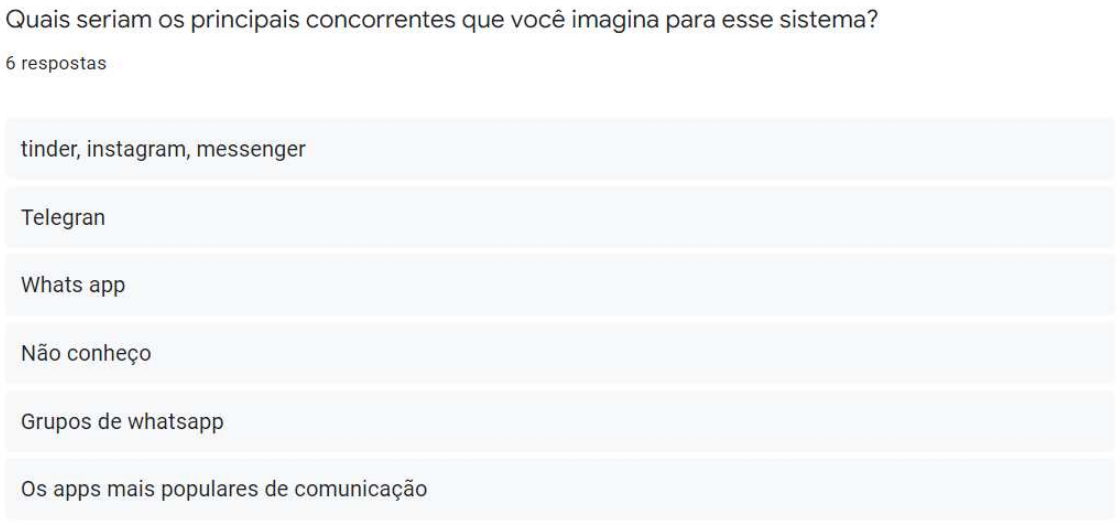

Figura 34 – Resposta 7

Fonte: Google Forms.

Figura 35 – Resposta 8

Fonte: Google Forms.

6.2 Anexo B - Publicação na Loja de Aplicativos Android - Google Play

Para ter acesso ao *download* do aplicativo é necessário estar em uma lista de testes e obter o *link*, como posteriomente o poderá mudar o endereço, não será anexado.

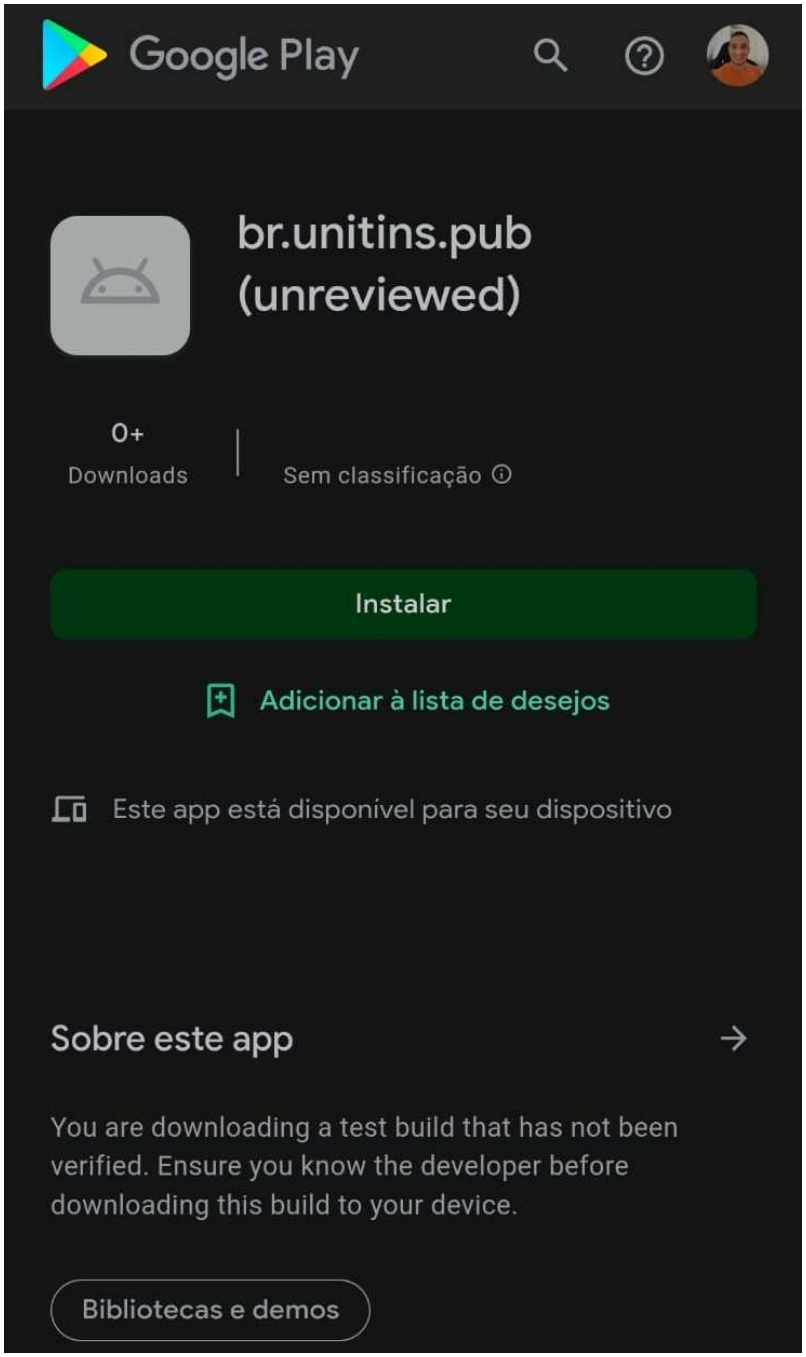

Figura 36 – Sistema desenvolvido publicado na loja de aplicativos Google Play.

Fonte: Google Play (2022).

7 Apêndices

7.1 Requisitos de Sistemas

7.1.1 Requisitos Funcionais

Requisitos funcionais descrevem as funções que o sistema deverá realizar. Sendo detalhadas para entendimento do funcionamento do sistema como atores, pré-requisitos, referências e prioridades Pádua (2003).

Abaixo segue os requisitos funcionais do sistema.

RF0001 Cadastrar Conta de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: -.
Descrição: Operação realizada pelo usuário para cadastro no sistema, e os seguintes campos devem ser preenchidos: Nickname, gênero, ícone e idade. Todos os campos devem ser preenchidos e os dados serão armazenados somente internamente no dispositivo.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0002 Editar Conta de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: RF0001.
Descrição: Operação realizada pelo usuário para alterar os dados dos campos referentes ao cadastro: Nickname, gênero, idade, humor e ícone.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0003 Excluir Conta de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde o mesmo excluirá todos os dados do seu perfil do armazenamento interno do dispositivo.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0004 Bloquear Conta de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: RF0001, RF0010.
Descrição: Operação realizada pelo usuário e bloqueará o acesso a um chat privado com algum usuário específico, desabilitando a função do mesmo de solicitar um chat.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0005 Desativar Conta de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: RF0001.
Descrição: Operação realizada pelo usuário, onde o mesmo atualizará o status da conta para inativo.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0006 Votar para banir de Usuário

Atores: Usuário.
Caso de Uso: Gerenciar Usuários.
Referência: RF0001, RF0010.
Descrição: Operação realizada pelos usuários para banir algum usuário específico da sala devido à má conduta.
Prioridade: ☐ Essencial ☒ Importante ☐ Desejável

RF0007 Entrar em Sala Pública

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde é possível a utilização de uma sala pública com outros usuários, sendo obrigatório o mesmo estar cadastrado.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0008 Buscar Salas Públicas

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde é possível a pesquisa de uma ou mais salas públicas.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0009 Visualizar Salas Públicas em Mapa

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde é possível visualizar as salas públicas em formato de mapa.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0010 Visualizar Integrantes da Sala

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001, RF0007.
Descrição: Operação realizada pelo usuário ao entrar na visualização dos integrantes da sala, podendo pesquisar por nickname.
Prioridade: ☐ Essencial ☒ Importante ☐ Desejável

RF0011 Buscar Salas Privadas

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde é possível a pesquisa de uma ou mais salas privadas.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0012 Solicitar Sala Privada

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001, RF0010.
Descrição: Operação realizada pelo usuário para solicitar a criação de uma sala privada com o usuário requerido, onde é obrigatório aguardar pela resposta do mesmo.
Prioridade: ☐ Essencial ☒ Importante ☐ Desejável

RF0013 **Entrar em Sala Privada**

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001, RF0012.
Descrição: Operação realizada pelo usuário onde é possível a utilização de uma sala privada com outro usuário, sendo obrigatório o mesmo estar cadastrado, e ter a solicitação de criação de sala privada aceita pelo usuário requerido.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0014 **Excluir Sala Privada**

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde o mesmo excluirá a sala privada e todos os dados do armazenamento interno do dispositivo.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0015 **Silenciar Sala**

Atores: Usuário.
Caso de Uso: Funcionalidades.
Referência: RF0001.
Descrição: Operação realizada pelo usuário onde o mesmo poderá silenciar qualquer sala privada ou pública.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RF0016 **Compartilhar Aplicativo**

Atores: Usuário e Empresa.
Referência: RF0001 (Usuário).
Caso de Uso: Funcionalidades.
Descrição: Operação realizada pelo usuário e pelo representante da empresa, sendo possível compartilhar o aplicativo com outros usuários em aplicativos de terceiros.
Prioridade: ☐ Essencial ☒ Importante ☐ Desejável

7.1.2 Requisitos Não-Funcionais

Requisitos não-funcionais são os aspectos necessários para garantir a qualidade do sistema como requisitos de desempenho, requisitos de persistência, e requisitos técnicos Pádua (2003).

Abaixo segue os requisitos não-funcionais do sistema.

RNF0001 Linguagem de Programação para Front-End

Descrição: O front-end deverá ser implementado usando a linguagem de programação Dart com o framework Flutter.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0002 Linguagem de Programação para Back-End

Descrição: A camada de Integração deverá ser implementada usando a linguagem de programação Javascript (utilizando a interface "Typescript") com o framework Node.js.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0003 Requisito de Disponibilidade

Descrição: O sistema necessita de GPS e internet ligados a todo momento enquanto o usuário permanecer no local, para monitoramento da localização em tempo real, entrega e sincronização das mensagens e serviços, necessitando da disponibilização em tempo integral de serviços de geolocalização da empresa TOMTOM e dos serviços de hospedagem em nuvem da Heroku.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0004 Requisito de Usabilidade

Descrição: O sistema deve utilizar interface intuitiva que facilita a utilização de variados tipos de usuários.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0005 Requisito de Disponibilidade de utilização do Front-End

Descrição: O front-end deve ser disponível para dispositivos móveis android e iOS.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0006 Requisito de hospedagem em nuvem

Descrição: O serviço da camada de integração, o back-end e o gerenciamento de base de dados devem estar hospedados e disponibilizados na plataforma heroku.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável

RNF0007 Requisito de Arquitetura

Descrição: O sistema deve funcionar na estrutura de arquitetura REST, sendo todas as informações compartilhadas por requisições, sendo o front-end em flutter responsável por estabelecer a comunicação entre os usuários, e o back-end em node.js encarregado da sincronização de mensagens, e o gerenciamento das salas criadas por geolocalização através da API da TOMTOM.
Prioridade: ☒ Essencial ☐ Importante ☐ Desejável