Talk to Refugee (T2R) - Chat de Integração para Imigrantes e Refugiados

Lucas Vinícius Carmassi, Daniela Marques

Análise e Desenvolvimento de Sistemas - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Campus Hortolândia
Hortolândia – SP – Brasil

lucas.carmassi@aluno.ifsp.edu.br, marquesdaniela@ifsp.edu.br

Abstract. *Interculturality and cultural diversity are essential factors for social integration in the context of refugees and immigrants. In this scenario, technological solutions stand out as fundamental tools for information exchange and mutual support. This work proposes the development of a communication platform aimed at the inclusion and reintegration of refugees in Brazil, connecting them with social groups and other established immigrants. The solution consists of a real-time chat system, facilitating users' adaptation to a new community through potential support networks formed on the platform.*

Resumo. *A interculturalidade e a diversidade cultural são fatores essenciais para a integração social no contexto de refugiados e imigrantes. Nesse cenário, soluções tecnológicas se destacam como ferramentas fundamentais para a troca de informações e o suporte mútuo. Este trabalho propõe o desenvolvimento de uma plataforma de comunicação voltada para a inclusão e reintegração de refugiados no Brasil, conectando-os a grupos sociais e outros imigrantes já estabelecidos. A solução consiste em um sistema de chat em tempo real, facilitando o processo de adaptação dos usuários a uma nova comunidade por meio de possíveis redes de apoio formadas na plataforma.*

1. Introdução

Em 1950 surge o ACNUR (Alto Comissariado das Nações Unidas para Refugiados), a agência da ONU para refugiados, com intuito de reassentar refugiados europeus que estavam sem um lar após a Segunda Guerra Mundial. Em 1967 a atuação da agência passa pelas fronteiras europeias e sua atuação começa a ser global, seu principal objetivo é garantir o direito básico dos refugiados. Nas últimas décadas, o número de refugiados em todo o mundo atingiu proporções proeminentes, mais de 67 milhões de pessoas, segundo estimativas realizadas pela agência. No Brasil, o número de refugiados cresce cada vez mais, e estima-se que, em outubro de 2018, segundo Teixeira (2021), havia cerca de 10 mil refugiados no país. Esse número cresceu para cerca de 32 mil no ano de 2019, um aumento de aproximadamente 22 mil pessoas no período de 1 ano; evidenciando a necessidade de olhar para este assunto. De acordo com o Ministério da Justiça e Segurança Pública (2025), por meio do relatório Refúgio em Números, que traz estatísticas atuais e o perfil das populações refugiadas, solicitantes de asilo, apátridas e migrantes no Brasil, o número de reconhecimentos de refugiados no país em 2024 é de cerca de 77 mil. No mundo, mais de 120 milhões de pessoas foram deslocadas à força de suas comunidades de origem entre junho de 2023 e junho de 2024.

> Refugiados são pessoas que estão fora de seu país de origem devido a fundados temores de perseguição relacionados a questões de raça, religião, nacionalidade,

> pertencimento a um determinado grupo social ou opinião política, como também devido à grave e generalizada violação de direitos humanos e conflitos armados. (ACNUR, 2024)

A obtenção de documentos necessários para a normalização da estadia do refugiado no país receptor é um dos primeiros passos para sua inserção. Esse processo frequentemente resulta em dificuldades para conseguir uma moradia, prosseguir com os estudos e até ser inserido no mercado de trabalho. Outra barreira para a integração do refugiado que se destaca: a diferença entre o sistema educacional e a diferença de idioma do Brasil com o do país de origem, resultando em uma sensação de isolamento e dificuldade de convivência. Discriminação e preconceito é também uma das barreiras que os refugiados enfrentam; tendo que lidar com hostilidade; marginalização; racismo e misoginia. Além da falta de recursos financeiros e emprego, que são pontos críticos e de extrema importância; muitas vezes o refugiado chega ao país acolhedor com uma difícil situação econômica, precisando de apoio financeiro ou um trabalho, fazendo com que sejam forçados a aceitar empregos que não exigem qualificações, mesmo possuindo mestrado, treinamento ou experiência, para poder garantir sua sobrevivência (TEIXEIRA, 2021).

Apesar dos trabalhos realizados pela ACNUR em prol da integração de um refugiado em uma nova comunidade, ainda existem muitas barreiras para que de fato sua re-inserção à longo prazo ocorra com sucesso. O refúgio em um novo país envolve superar barreiras culturais, políticas, acadêmicas e linguísticas, complicando ainda mais o processo de adaptação e integração dos refugiados. Nesse contexto, ferramentas tecnológicas podem desempenhar um papel crucial na criação de uma rede de apoio para o refugiado. Segundo Ramos (2009), a interculturalidade e a diversidade cultural são fundamentais para promover a troca de experiências e criar laços comunitários. Assim, uma plataforma digital pode conectar refugiados a ONGs, grupos sociais e membros da comunidade, possibilitando o suporte para sua adaptação. O aprendizado da língua do país de acolhimento é essencial; um software de comunicação não substitui o ensino formal do idioma, mas pode auxiliar ao permitir interações mais naturais e funcionais. A troca de experiências e informações viabilizadas por um sistema de informação focado na comunicação pode ajudar a diminuir o isolamento, criar uma rede de apoio e facilitar o processo de adaptação, proporcionando uma inserção duradoura em uma nova comunidade.

O presente trabalho propõe documentar o processo de construção do protótipo *Talk to Refugee*, um sistema para trocas de mensagens, visando conectar os refugiados recém-chegados no Brasil com outros imigrantes já integrados; colaboradores; organizações não governamentais e grupos de movimentos sociais. A intenção é promover facilidade no processo de reintegração do refugiado por meio do contato com essa comunidade de apoio, promovendo uma experiência acolhedora e inclusiva, com uma interface de usuário acessível, eficiente e de estética agradável. Para isso, foram utilizados processos e artefatos da área de Engenharia de Software, a análise de trabalhos correlatos para documentação, compreensão da expectativa de funcionamento do sistema e identificação dos principais pontos do desenvolvimento.

2. Trabalhos Correlatos

A Seção de trabalhos correlatos tem como objetivo identificar funcionalidades e ferramentas específicas referentes a sistemas de chat, tendo como ponto de partida outros sistemas com ideias semelhantes, possibilitando a comunicação entre usuários de forma segura e eficiente. Sendo esta uma revisão essencial para compreensão do contexto da aplicação que foi desenvolvida e as

contribuições do presente estudo.

2.1. Rakuten Viber

É uma aplicação para trocas de mensagens de texto e videochamadas, unindo pessoas do mundo todo em uma plataforma que se destaca por sua segurança, com uma comunicação encriptografada de ponto a ponto (VIBER, 2024). A Figura 1 ilustra como é a página inicial e principal do sistema.

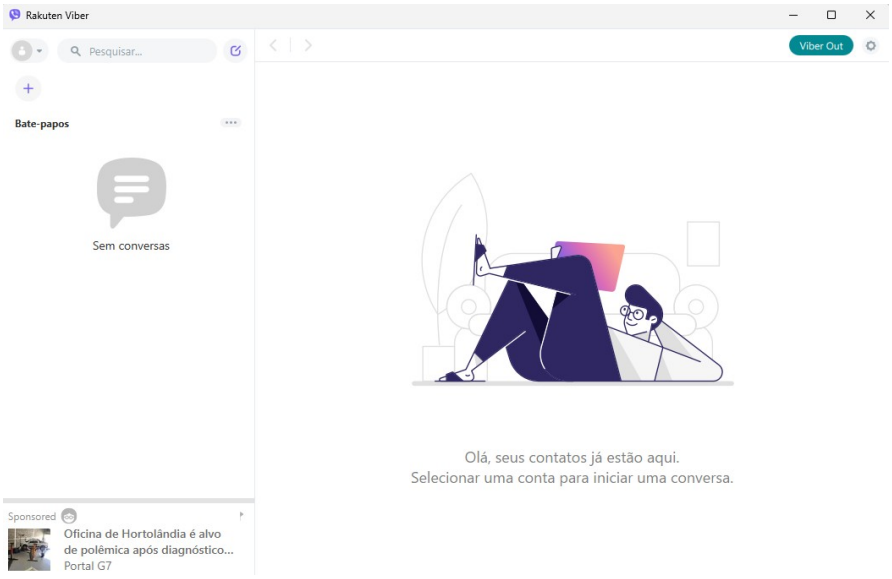


Figura 1. Página principal do sistema Rakuten Viber

2.2. Signal

O sistema foi desenvolvido e é mantido pela organização sem fins lucrativos Signal desde 2013. A aplicação tem como foco principal a segurança e privacidade das informações do usuário, tendo um protocolo de segurança de código aberto próprio (SIGNAL FOUNDATION, 2024). A Figura 2 ilustra como é a página inicial e principal do sistema.

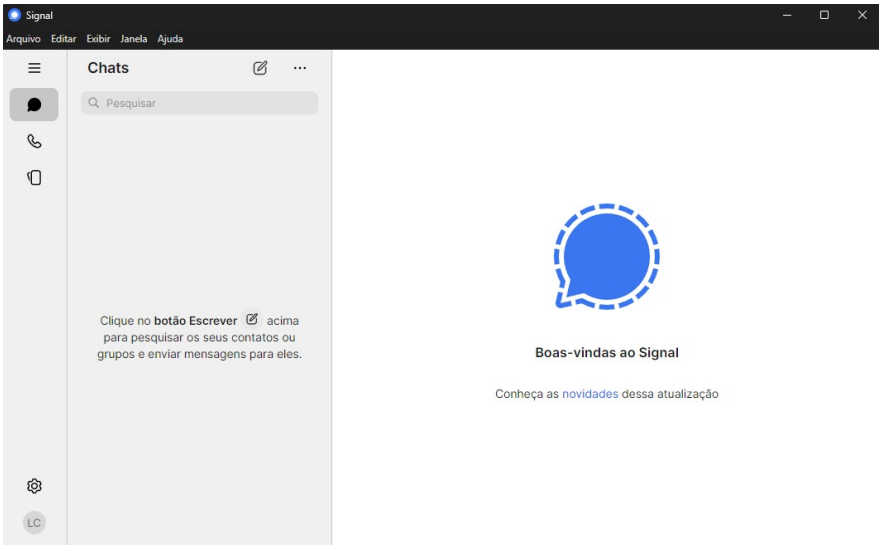


Figura 2. Página principal do sistema Signal

2.3. Comparação

A análise dos trabalhos correlatos, como Signal e Rakuten Viber, trouxeram compreensões sobre as funcionalidades necessárias e esperadas pelos usuários. Foram selecionadas as principais funcionalidades a serem adotadas pelo projeto Talk To Refugee (T2R), como listado na Tabela 1.

Funcionalidades	Rakuten Viber	Signal Messenger	T2R
Conversas privadas	✔	✔	✔
Conversas em grupo	✔	✔	✔
Chamadas de vídeo	✔	✔	✖
Multiplataforma	✔	✔	✔
Busca por localização	✖	✖	✔
Tradução de conversa	✖	✖	✔
Tradução do sistema	✔	✔	✔

Tabela 1. Comparação trabalhos correlatos com T2R

3. Referencial Teórico

Na presente seção serão detalhados os principais recursos tecnológicos e metodologias aplicadas na construção do sistema T2R, referenciando a utilização de cada um dos tópicos e explicando a sua importância para obtenção do resultado final do sistema.

3.1. API Restful

As APIs Restful são implementações de sistemas distribuídos que estão em conformidade com o estilo de arquitetura REST (*Representational State Transfer*) e utilizam a comunicação com os conjuntos de definições e protocolos API (*Application Programming Interface*) . Em suma, uma API Restful é a implementação de um conjunto de definições e protocolos que propõe uma maneira facilitada de comunicação entre sistemas, permitindo que um consumidor faça requisições padronizadas e receba respostas padronizadas. Ela funciona como um mediador entre usuários e serviços web, fornecendo acesso a recursos e informações de forma segura e controlada, simplificam a integração sem exigir que o usuário conheça detalhes internos, como armazenamento ou origem dos dados (RED HAT, 2024a).

3.2. Conteinerização

A conteinerização consiste no empacotamento de aplicações com todas as suas dependências de forma totalmente separada da infraestrutura da equipe de desenvolvimento em *containers*, possibilitando a execução de aplicações em qualquer ambiente que tenha algum mecanismo de

conteinerização instalado, mitigando problemas de incompatibilidade de infraestrutura (RED HAT, 2024b).

Segundo a Amazon Web Services (2023), existem diversas vantagens em se utilizar a tecnologia de conteinerização no desenvolvimento e entrega de software. Entrega portabilidade ao possibilitar que uma aplicação seja criada em uma máquina e replicada em diferentes sistemas operacionais que suportem essa tecnologia. Escalabilidade, pois *containers* são leves e de rápida inicialização, permitindo múltiplas execuções em uma única máquina. Tolerância a falhas, como os *containers* executam de forma isolada, uma falha não afeta as demais dependências do sistema. Por fim, agilidade, pois *containers* permitem a solução de problemas e alterações de código fonte sem impactar o sistema operacional ou outros serviços, acelerando os ciclos de lançamento de novas versões.

3.3. Prototipação

Um protótipo é uma prática da Engenharia de Software que consiste em desenvolver uma versão primária para um sistema de software, tem como principais características: demonstração de conceitos, experimentação de opções e descoberta precoce de mudanças; problemas e suas soluções. A prototipação permite que essa primeira versão possibilite a identificação de pontos fortes e fracos resultando em novas ideias de requisitos, além de revelar erros e omissões em requisitos do sistema por meio da utilização do protótipo, trazendo clareza sobre a viabilidade do produto. Outra característica é que a prototipação é a única maneira coerente de se desenvolver interfaces junto do usuário, isso porque diagramas representativos não trazem uma experiência fidedigna da experiência de uso do sistema com o usuário (SOMMERVILLE, 2015, p.47).

3.4. Mensageria

O desenvolvimento de sistemas distribuídos vem acompanhado da necessidade da escolha de uma comunicação que se enquadre nos requisitos do sistema, para o sistema T2R há a necessidade da escolha de uma comunicação que atenda a necessidade de reatividade do envio de mensagens no chat para possibilitar uma conversa em tempo real entre os usuários.

Segundo Arruda e Martins (2023), a mensageria é a camada de comunicação responsável pelo gerenciamento de mensagens enviadas entre os sistemas em um contexto de sistemas distribuídos, essa camada tem grande responsabilidade no funcionamento dos sistemas que estão conectados por ela, uma vez que toda a integração dessa comunicação depende da mesma, além de ter que lidar com outras questões como concorrência e indisponibilidade.

4. Metodologia

A Seção Metodologia apresenta a abordagem utilizada para criação do sistema T2R. A primeira atividade começou com o levantamento dos requisitos construídos para compreensão das funcionalidades do sistema e entendimento do escopo, sendo gerado o diagrama de caso de uso e diagrama de classe, os artefatos foram elaborados com base no que foi analisado nos trabalhos correlatos.

Após essa primeira análise foi desenvolvida a modelagem física da base de dados para

compreender como os dados serão persistidos. Com isso, pode-se iniciar o desenvolvimento incremental levando em consideração os requisitos já definidos, permitindo uma implementação controlada e testes contínuos.

A aplicação *back-end* será constituída por uma API Restful para registrar as informações necessárias para o gerenciamento das informações dos usuários, criação e envio de mensagens de chat; e uma aplicação *front-end* reativa para utilização das funcionalidades. Por fim, as aplicações terão suas dependências encapsuladas e gerenciadas por uma *engine* de administração de *container*, isolando o ambiente para facilitação de implantação e execução das aplicações.

5. Desenvolvimento

Considerando a compreensão dos recursos tecnológicos explorados nas seções anteriores, a presente seção tem como objetivo detalhar as etapas de construção das funcionalidades do sistema.

5.1. Compreendimento do Sistema

Antes de começar a implementação do sistema em si, primeiro foi necessário entender quais e como seriam as funcionalidades deste sistema, iniciando pela análise e entendimento de sistemas do mesmo nicho que o T2R como descrito na Seção 2. Com isso, foi possível iniciar o desenvolvimento dos artefatos da Engenharia de Requisitos relacionados a modelagem do sistema que são: diagrama de caso de uso, diagrama de classe e modelagem física da base.

5.1.1 Engenharia de Requisitos

Nesta seção serão apresentados três diagramas: o diagrama de caso de uso, que ilustra as interações entre os atores e o sistema; o diagrama de classes, que define a estrutura e os relacionamentos das entidades; e a modelagem física do banco de dados, que detalha como os dados serão persistidos e como irão interagir no sistema.

Segundo Sommerville (2015), a Engenharia de Requisitos é o processo de descoberta, análise, documentação, conferência e restrição de requisitos; requisitos são descrições dos serviços de negócio que o sistema deve oferecer e refletem as necessidades dos usuários de um sistema que atende a um propósito específico, assim como suas restrições. Existem dois tipos de requisitos: os funcionais e os não funcionais. Requisitos funcionais são declarações dos serviços que o sistema deve oferecer, de como os mesmos devem se comportar de acordo com diferentes entradas e estados, um ponto importante a se considerar é que os requisitos funcionais, além de declarar o que o sistema deve fazer, também declaram o que sistema não deve fazer, ou seja, declarar seus limites.

A autenticação do usuário se baseia na disponibilização de uma funcionalidade de *login* onde o usuário poderá acessar a plataforma com seu e-mail e senha, sendo redirecionado para a página principal da aplicação em caso de sucesso ou mantido na página de autenticação com a mensagem de erro respectiva ao motivo da falha ao acessar a plataforma. O usuário poderá se cadastrar na plataforma por meio de uma página de cadastro preenchendo informações como: nome, descrição, e-mail, senha, data de nascimento, cidade e estado atual, e tipo de usuário. Por

fim, os usuários que estão autenticados no sistema devem poder se desconectar da plataforma, tendo seu token de autenticação removido do navegador usado pelo usuário. Além disso, o sistema deve possibilitar o gerenciamento do perfil, permitindo que os usuários visualizem e alterem suas informações, modifiquem a senha, alterem sua foto de perfil ou desativem sua conta. Deve também fornecer uma funcionalidade de busca de interlocutores, permitindo pesquisar por localização, tipo de perfil ou nome, além de iniciar conversas e criar grupos a partir dos resultados da busca.

Ademais, deve ser possível a interação por meio de conversas privadas, permitindo que os usuários listem as conversas existentes, visualizem mensagens enviadas, enviem novas mensagens, responda a mensagens específicas, enviem arquivos e denunciem mensagens. As conversas em grupo também devem ser suportadas, permitindo listar os grupos em que os usuários participam, enviar mensagens com possibilidade de responder a mensagens específicas, alterar informações e fotos dos grupos, sair de grupos e denunciar mensagens. Por fim, o sistema deve oferecer funcionalidades para moderação da plataforma, permitindo que administradores acessem o sistema com credenciais previamente cadastradas, listem perfis denunciados com as mensagens associadas, desativem contas de interlocutores.

5.1.2 Requisitos Não Funcionais

Segundo Sommerville (2015), os requisitos não funcionais por sua vez são restrições sobre as funcionalidades oferecidas pelo sistema, restrições como: tempo de resposta; processo de desenvolvimento; e padrões. Geralmente as restrições declaradas pelos requisitos não funcionais se aplicam ao sistema como um todo.

Em relação ao desempenho, é imprescindível que toda requisição tenha um tempo máximo de resposta de 1 segundo, assegurando uma experiência ágil para os usuários. Além disso, o sistema deve ser escalável, possibilitando o suporte ao crescimento contínuo tanto no número de usuários quanto no volume de dados sem comprometer sua eficiência. No quesito segurança, o sistema deve proteger os dados dos usuários, implementando políticas de autenticação seguras e evitando a exposição de informações sigilosas, para estar em conformidade com a Lei Geral de Proteção de Dados (BRASIL, 2025), solicitando apenas dados sensíveis que tenham valor direto para o usuário e garantindo que estes sejam utilizados de forma responsável.

No que diz respeito à portabilidade, o sistema deve funcionar de maneira compatível com diferentes dispositivos, sistemas operacionais e navegadores, garantindo uma experiência uniforme e acessível para os usuários, independentemente da plataforma utilizada. Para facilitar a observabilidade, *loggings* devem ser implementados, permitindo acompanhar o fluxo de execução e analisar possíveis erros. Informações métricas também devem estar disponíveis para monitorar o desempenho da aplicação e identificar pontos de melhoria.

Por fim, para assegurar a manutenibilidade, o sistema deve ter um código modular que facilite a adição de novas funcionalidades, a refatoração e a manutenção das já existentes. Além disso, um mínimo de 75% de cobertura de testes deve ser garantido, utilizando testes unitários para verificar o correto funcionamento de cada unidade e testes de integração para validar os fluxos da aplicação como um todo.

5.1.3 Diagrama de Caso de Uso

Segundo Sommerville (2015), o diagrama de caso de uso é um tipo de diagrama para apoiar a modelagem dos casos de uso, sendo o caso de uso uma descrição simples da intenção do usuário nessa interação com o sistema. A Figura 3 ilustra quais as interações dos atores e sistema, também é possível ver a atribuição de funcionalidade baseado nos contextos definidos no documento de Levantamento de Requisitos (Apêndice 1).

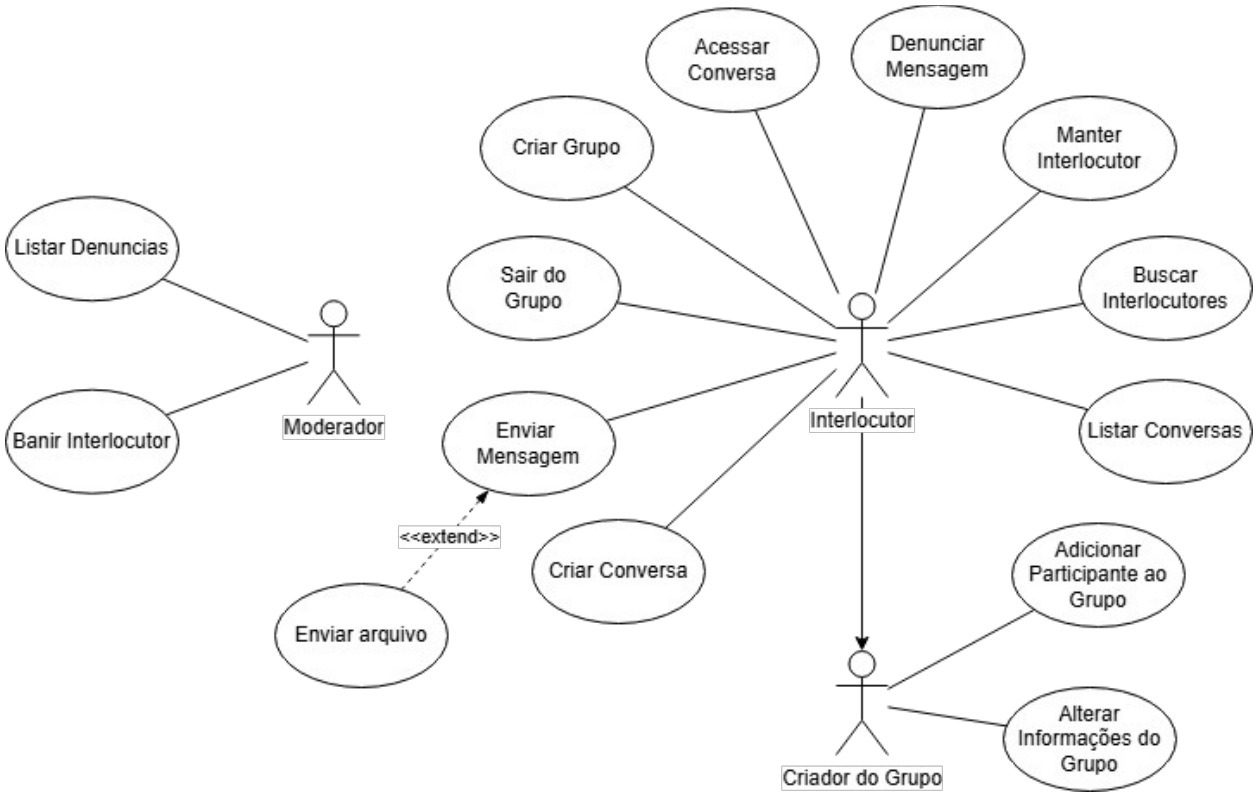


Figura 3. Diagrama de caso de uso - atores e suas intenções

5.1.4 Diagrama de Classes

Segundo Sommerville (2015), o diagrama de classes é utilizado em sistemas orientados a objetos para demonstrar quais são e como as classes se relacionam, uma classe por sua vez é uma definição geral de um tipo de objeto do sistema.

A Figura 4 ilustra como foi estruturado o sistema para atender os requisitos definidos anteriormente, permitindo: troca de mensagens em conversas privadas ou em grupos; busca de interlocutores; e sistema de moderação da plataforma. Representadas pelas seguintes classes: (i) Interlocutor, define os usuários da plataforma que podem ser de diferentes perfis conforme indicado pelo enum InterlocutorPerfil; (ii) Conversa, responsável pela interação dos interlocutores representando a troca de mensagens entre exclusivamente dois usuários como pode-se ver pela multiplicidade entre Interlocutor e Conversa; (iii) Grupo com a mesmo intuito da classe Conversa, porém múltiplos usuários podem se comunicar por meio dela; (iv) Mensagem, responsável pelo que foi enviado pelos interlocutores em uma conversa ou em um

grupo, podendo ser desde texto até arquivos como imagens; (v) Denúncia, como forma de moderação, permite que um usuário reporte conteúdos considerados indevidos, associando a denúncia diretamente a uma mensagem específica; e, (vi) Moderador, que é responsável por gerenciar as denúncias feitas pelos usuários, sendo possível desativar contas de interlocutores quando necessário.

Os relacionamentos entre as classes refletem o funcionamento do sistema, como a necessidade de um grupo ter pelo menos dois interlocutores para ser criado, ou cada denúncia estar sempre associada a uma única mensagem, garantindo a rastreabilidade da moderação.

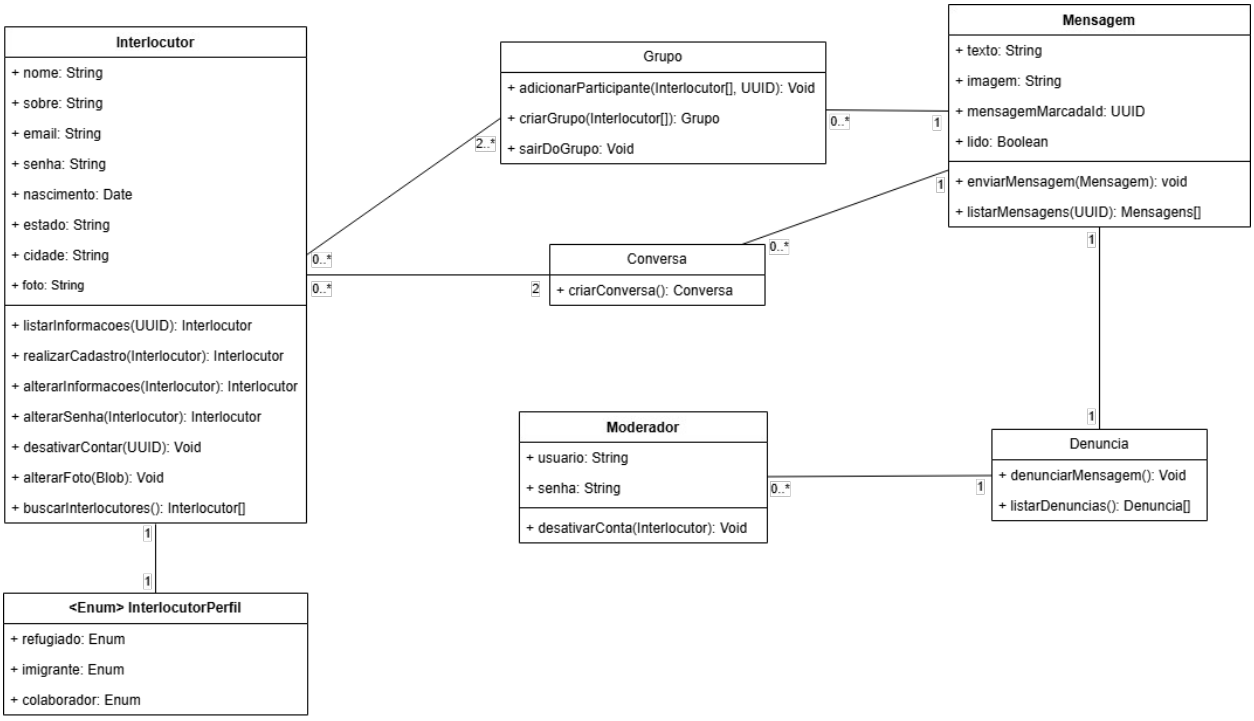


Figura 4. Diagrama de classes - objetos do sistema T2R e como se relacionam

5.1.5 Modelagem Física do Banco de Dados

Segundo a Amazon Web Services (2024), a modelagem física de banco de dados é uma etapa na modelagem de uma base de dados onde os elementos são detalhados para refletir a implementação em uma plataforma específica de banco de dados. Ela inclui informações como: tipos de dados; restrições de nomenclatura; acionadores; procedimentos armazenados; e outros atributos que garantem a compatibilidade com o ambiente físico de execução.

Esse modelo é essencial para que administradores e desenvolvedores configurem adequadamente a base de dados em um projeto de software, seguindo as particularidades e limitações técnicas da plataforma escolhida. No caso do sistema T2R a plataforma escolhida para o armazenamento dos dados da aplicação foi o PostreSQL, que é um sistema gerenciador de banco de dados objeto-relacional de código aberto que combina confiabilidade, extensibilidade e suporte a *workloads* complexos. É amplamente utilizado por ser gratuito, extensível e por oferecer recursos que facilitam o desenvolvimento de aplicações e a administração de ambientes

tolerantes a falhas. Ele permite a criação de tipos de dados personalizados, funções específicas e a execução de código em diferentes linguagens de programação (POSTGRESQL, 2023).

A Figura 5 representa o diagrama da modelagem física da base de dados da aplicação, tem como objetivo detalhar os dados de cada entidade do sistema, seus tipos e como cada entidade se relaciona. Vale ressaltar que a modelagem física da base foi feita levando em consideração os incrementos que foram desenvolvidos no presente trabalho e não do sistema completo, uma vez que cada incremento adicionava novas entidades e relacionamentos na base de dados.

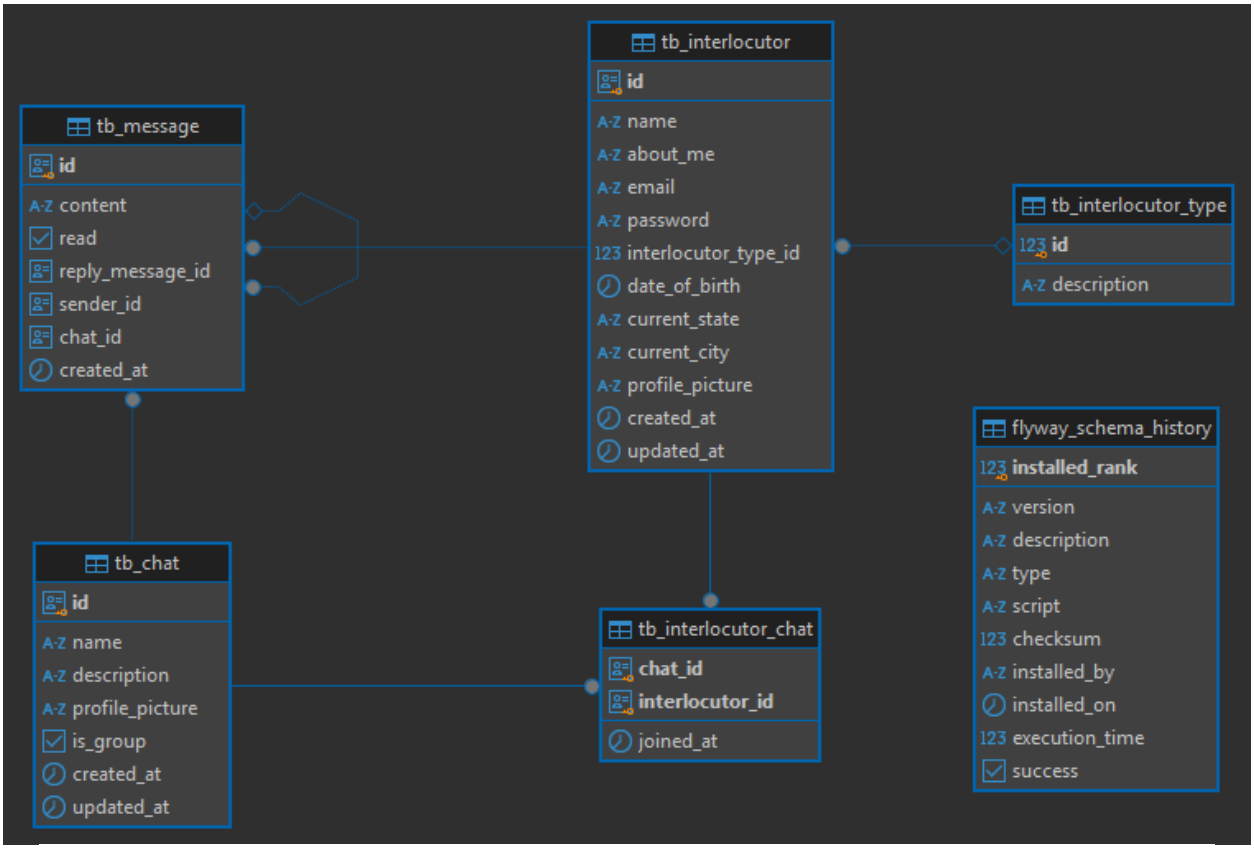


Figura 5. Modelagem física da base de dados - entidade e seus dados, e como se relacionam

5.2 Incrementos

O desenvolvimento baseado em incrementos consiste na ideia de quebrar o sistema completo em diversas entregas, sendo um desenvolvimento cíclico que tem como objetivo concluir o projeto final ao unir todas as versões disponibilizadas por cada incremento. Sendo assim, o desenvolvimento incremental consiste na ideia de desenvolver uma implementação inicial do sistema; obter *feedback* por meio da versão inicial ou incremento; e fazer o software evoluir de várias versões até o sistema final.

5.2.1 Primeiro Incremento: Criação dos projetos e Cadastro de Usuário

No primeiro incremento foi feita a criação e configuração dos projetos *back-end* e *front-end*,

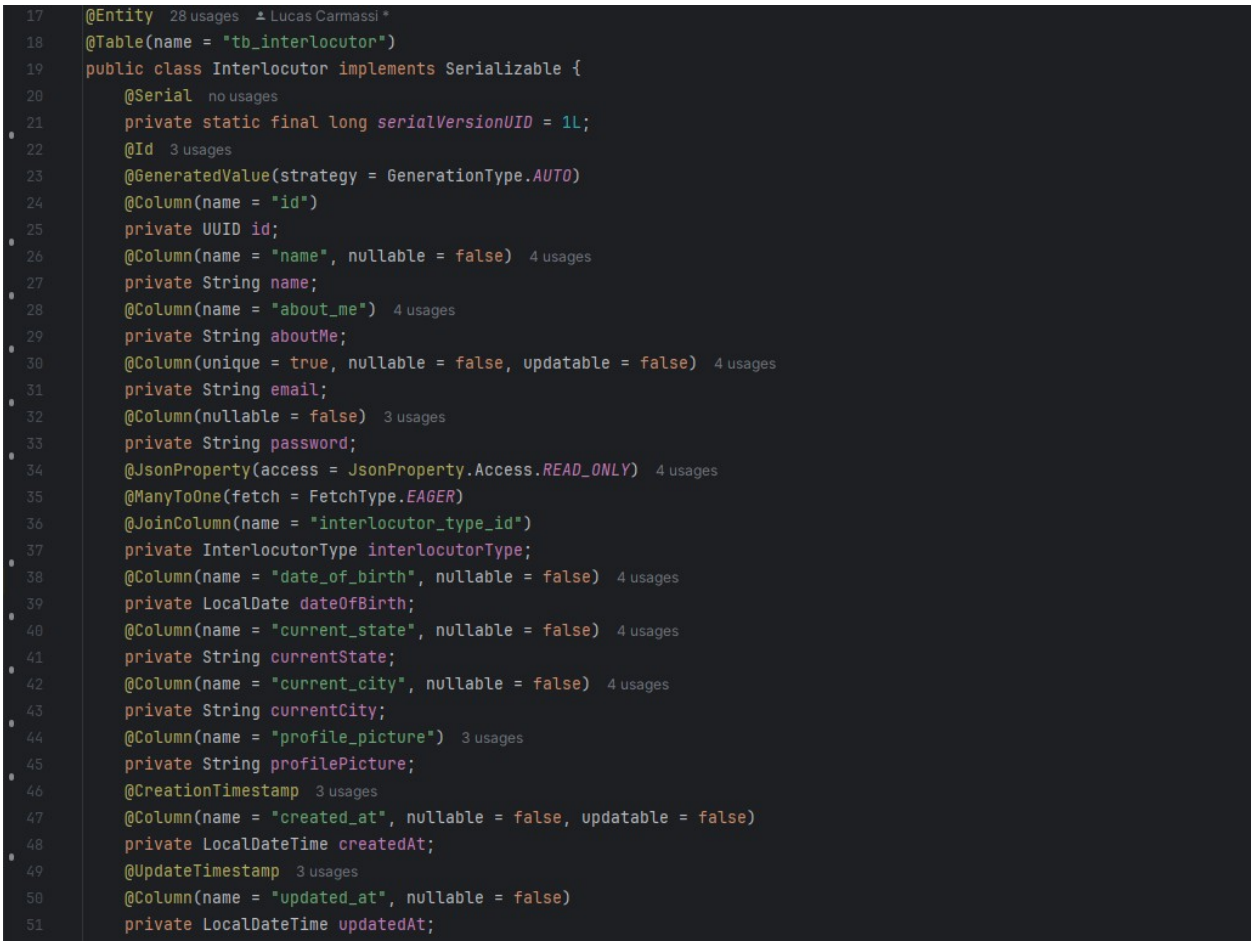
além da implementação da funcionalidade de cadastro de usuário. Iniciando com o projeto *back-end* com a disponibilização do *endpoint* referente ao requisito de criação de usuário e posteriormente a interface de usuário para a integração com o *endpoint* disponibilizado. A criação do projeto *back-end* foi feito pela plataforma Spring Initializr, uma ferramenta que auxilia na criação rápida de projetos Java Spring Boot configurados com as dependências necessárias, gera um projeto base levando em consideração as escolhas técnicas feitas, como: linguagem; versão; e dependências.

Após abrir o projeto na IDE, foi preciso fazer a configuração da conexão com a base de dados devido à dependência spring-boot-starter-data-jpa que cria uma instância de conexão na inicialização do projeto, fazendo com que seja necessário configurar a conexão antes mesmo de usar. Um *container* Docker foi utilizado para o banco de dados usando por meio do Docker Compose, ele define um serviço que utiliza a imagem oficial do PostgreSQL, mapeia a porta 5432 para acesso local e cria um banco chamado inclusao_digital com usuário e senha.

Com isso, foi possível começar a implementação do sistema *back-end*, inicialmente com a criação da tabela para persistência das informações do usuário por meio de uma *migration* utilizando a dependência *flyway* via comandos SQL. Além disso, foi feito o mapeamento para um objeto entidade que será gerenciado pelo projeto spring-data-jpa. A entidade consiste não somente por atributos referentes aos dados pessoais do usuário, mas também por alguns atributos para a autenticação do usuário, como é o caso dos campos email (linha 31 da Figura 6) e *password* (linha 33 da Figura 6), além de alguns atributos para gerenciamento dos registros, como é o caso dos campos createdAt e updatedAt, como é possível observar na Figura 6 nas linhas 48 e 51 respectivamente.

Após essas configurações, foi possível iniciar a implementação das classes das camadas *controller* e *service*. Outras classes de apoio a esse caso de uso foram utilizadas, como um DTO (*Data Transfer Object*) e uma interface *repository* para definir os métodos de pesquisa que poderão ser executados pelo objeto entidade. A classe da camada *controller* tem como objetivo receber a requisição REST do método POST na rota “/interlocutor”, contendo os valores do usuário no corpo da requisição, que serão validados pelo projeto *starter-validation* do Spring, conforme as especificações feitas para cada campo do corpo da requisição, definidas por um record “CreateInterlocutorRequest”. Após a validação, os dados são direcionados para a classe da camada *service* da aplicação, que, por sua vez, é responsável por verificar se os dados recebidos na requisição já estão cadastrados no sistema e, caso contrário, realizar o *encode* da senha do usuário e criar o registro na base de dados.

Com a disponibilização do *endpoint* de cadastro de usuário, iniciou-se a configuração inicial do projeto *front-end* para o desenvolvimento da interface de usuário e a implementação do primeiro caso de uso do sistema. A criação do projeto em React foi realizada através do *console*, utilizando o comando “npm create vite@latest my-project -- --template react”. É possível observar pelo comando que o projeto utiliza o Vite, uma ferramenta de *build* focada em oferecer uma experiência de desenvolvimento mais rápida e eficiente. Com o projeto criado, foi realizada a instalação do Tailwind CSS, um *framework* de utilitários CSS que gera estilos personalizados por meio do escaneamento dos arquivos em busca de classes. A instalação foi feita com os comandos “npm install -D tailwindcss postcss autoprefixer” e “npx tailwindcss init -p”, além da configuração dos *paths* para indicar quais arquivos devem ser validados.



```java
@Entity  28 usages  Lucas Carmassi *
@Table(name = "tb_interlocutor")
public class Interlocutor implements Serializable {
    @Serial  no usages
    private static final long serialVersionUID = 1L;
    @Id  3 usages
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private UUID id;
    @Column(name = "name", nullable = false)  4 usages
    private String name;
    @Column(name = "about_me")  4 usages
    private String aboutMe;
    @Column(unique = true, nullable = false, updatable = false)  4 usages
    private String email;
    @Column(nullable = false)  3 usages
    private String password;
    @JsonProperty(access = JsonProperty.Access.READ_ONLY)  4 usages
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "interlocutor_type_id")
    private InterlocutorType interlocutorType;
    @Column(name = "date_of_birth", nullable = false)  4 usages
    private LocalDate dateOfBirth;
    @Column(name = "current_state", nullable = false)  4 usages
    private String currentState;
    @Column(name = "current_city", nullable = false)  4 usages
    private String currentCity;
    @Column(name = "profile_picture")  3 usages
    private String profilePicture;
    @CreationTimestamp  3 usages
    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @UpdateTimestamp  3 usages
    @Column(name = "updated_at", nullable = false)
    private LocalDateTime updatedAt;
```

Figura 6. Modelo de Entidade Interlocutor com JPA e Hibernate

Com essa configuração, foi possível iniciar o desenvolvimento dos componentes e páginas do projeto, começando pela página de cadastro de usuário. Essa página oferece uma interface para realizar a requisição REST ao *endpoint* fornecido *pelo back-end*, conforme ilustrado na Figura 7. Por fim, foi desenvolvido um *script* que recebe os valores preenchidos nos campos da página e, utilizando a biblioteca “axios”, faz uma requisição POST para a rota “/interlocutor” disponibilizada pelo sistema *back-end*.

5.2.2 Segundo Incremento: Autenticação do Usuário e Alteração de Informações

A autenticação do usuário consiste na disponibilização de um *endpoint* no sistema *back-end*, responsável por validar os dados de e-mail e senha recebidos no corpo da requisição REST do método POST na rota “/interlocutor/auth”. O sistema verifica se existe algum interlocutor com aquele e-mail e senha cadastrados na base de dados e, em caso positivo, retorna um JWT, que é um padrão usado para representar informações de forma segura entre duas partes, oferecendo métodos de *encode* e *decode* dessas informações, incluindo os dados do usuário autenticado. Em caso negativo, o sistema retorna uma mensagem de erro indicando que a autenticação falhou. Com a disponibilização desta rota, foi possível implementar uma interface de usuário para receber as informações de autenticação, com uma estética semelhante à tela de cadastro mencionada anteriormente, mas contendo apenas os campos de e-mail e senha. Além disso, foi

desenvolvido um *script* responsável por processar as informações de login do usuário e realizar a chamada ao *endpoint* correspondente a esse caso de uso. Também foi implementada uma âncora entre as telas de cadastro e autenticação, facilitando o usuário a alternar entre as duas páginas.

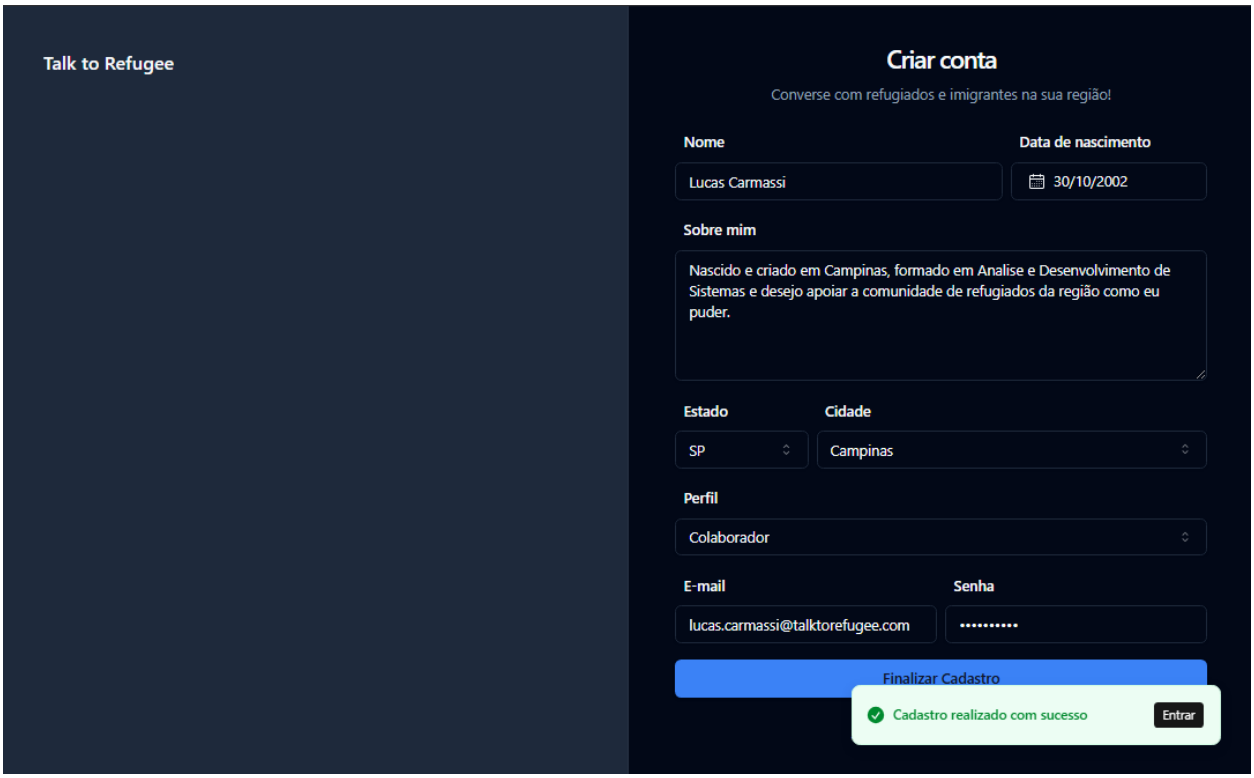


Figura 7. Interface de usuário para cadastro de interlocutores

A finalização deste caso de uso permitiu o desenvolvimento de outras funcionalidades que exigem a autenticação do usuário no sistema e o acesso às suas informações. Com isso, a alteração dos dados do usuário foi o próximo caso de uso implementado neste incremento. Vale destacar que a arquitetura detalhada no primeiro incremento, apresentada na Seção 5.2.1, também se aplica aos demais casos de uso. Assim, para o desenvolvimento deste caso de uso, seguiu-se a mesma abordagem documentada anteriormente: a disponibilização de um *endpoint* utilizando o método PUT para receber os dados a serem alterados, além da recuperação do JWT presente no header da requisição. A partir do JWT, foi extraído o UUID do usuário, utilizado como filtro para localizar o registro correto na base de dados. Dessa forma, as informações alteradas foram mapeadas para um objeto entidade, que foi então atualizado e salvo na base.

Além disso, foi desenvolvida uma interface de usuário para permitir a edição das informações do usuário. Essa interface apresenta campos para que os dados possam ser visualizados e atualizados conforme necessário, de forma alinhada ao restante do sistema. Para complementar, um *script* foi implementado para realizar a requisição ao *endpoint* deste caso de uso, garantindo que os dados alterados sejam enviados corretamente e que as respostas do sistema sejam processadas. A Figura 8 apresenta o *design* dessa interface.

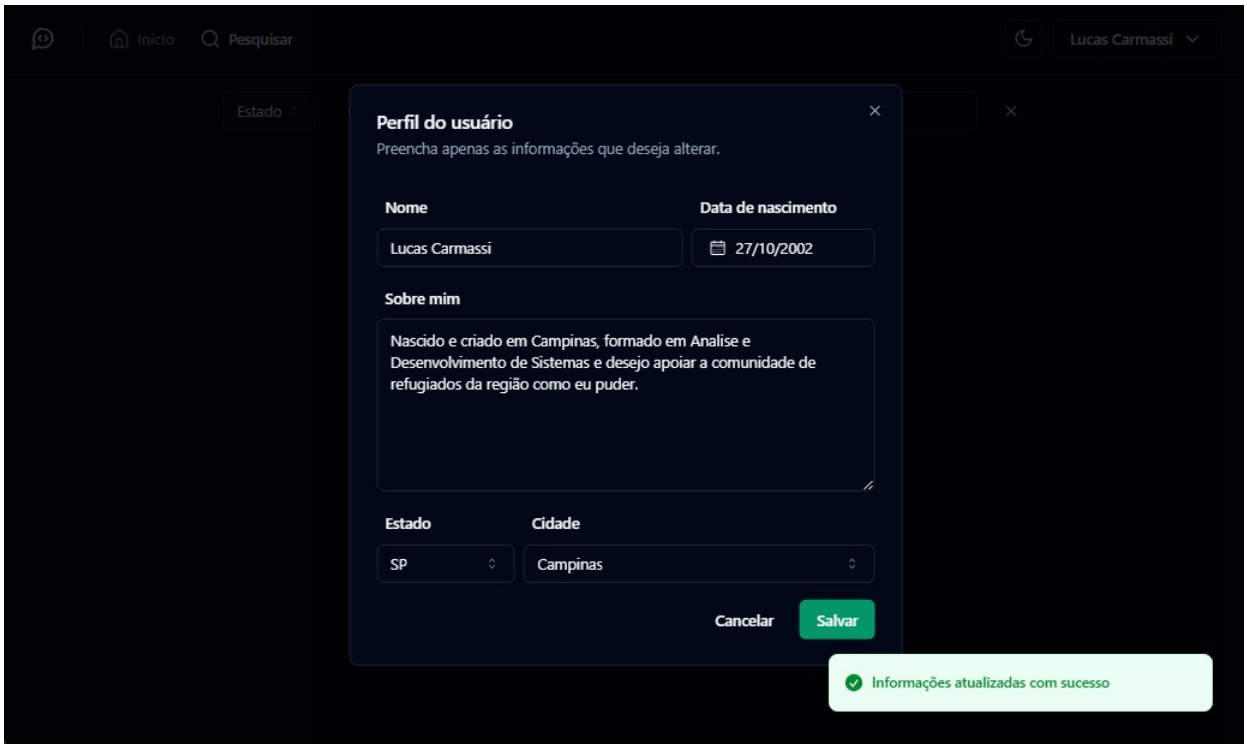


Figura 8. Interface de usuário para atualização de informações

5.2.3 Terceiro Incremento: Busca de Interlocutores

Com a disponibilização dos casos de uso de cadastro, autenticação e alteração de informações, o caso de uso seguinte deveria ser o de busca de interlocutores, uma vez que é necessário encontrar outro usuário antes de poder iniciar uma conversa com ele. Por isso, o terceiro incremento se inicia com a criação de um *endpoint* do tipo POST na rota "/interlocutor/search", que recebe em seu *body* três possíveis parâmetros para filtros: estado, cidade e nome. Os filtros são aplicados conforme forem preenchidos, tendo como padrão, caso nada seja preenchido, uma resposta sem nenhum filtro aplicado, nesse caso serão retornados todos os usuários cadastrados, exceto o usuário autenticado. Esse filtro é implementado por meio de uma consulta SQL que possui uma condicional em cada cláusula WHERE, no qual valida se o parâmetro é nulo ou não. Caso seja nulo, a consulta ignora essa cláusula, como é possível ver na Figura 9.

Após a disponibilização deste *endpoint*, o passo seguinte foi desenvolver uma interface de usuário para a listagem de interlocutores, incluindo campos para aplicação dos filtros na busca de interlocutores, seguindo os padrões estéticos já adotados para as demais interfaces. Por fim, foi implementado um *script* para realizar a requisição ao *endpoint* deste caso de uso, fazendo com que cada interação com os componentes de filtros execute uma requisição ao *back-end* e atualize a listagem de interlocutores, trazendo reatividade à interface.

5.2.4 Quarto Incremento: Criação de Conversas, Envio e Leitura de Mensagens

Os casos de uso de Criação de Conversa; Envio de Mensagem e Acessar Conversa compõem o quarto incremento e o funcionamento principal do sistema; a conversa privada entre os usuários.

A implementação deste incremento foi iniciada pela disponibilização do *endpoint* de método POST com a rota "/chats/message/send" para envio das mensagens, cadastrando cada mensagem na base de dados para atender a necessidade de histórico de conversas e conversas assíncronas. Além disso, será publicada uma mensagem em um tópico STOMP (*Simple Text Oriented Messaging Protocol*), que é um protocolo de mensageria leve para comunicação entre cliente e servidor que permitirá o sistema *back-end* se conectar a um *broker* de mensagem que será consumido pela aplicação *front-end*, possibilitando assim, a comunicação síncrona nas conversas entre os usuários. Como é possível ver na Figura 10, o método *send* é responsável pelo caso de uso de envio de mensagem, iniciando pela busca do chat anteriormente criado (linha 46), seguido por: a busca da mensagem referenciada caso exista (linhas 50 e 51); a relação do identificador do usuário autenticado com a mensagem (linhas 53 e 54); o cadastro da mensagem na base de dados (linha 56); e publicação da mensagem no tópico (linha 65).

```
@Query(  1 usage   Lucas Carmassi
        value = "select interlocutor.* " +
                "from tb_interlocutor interlocutor " +
                "where (:state is null or interlocutor.current_state ilike concat('%', :state, '%')) " +
                "and (:city is null or interlocutor.current_city ilike concat('%', :city, '%')) " +
                "and (:name is null or interlocutor.name ilike concat('%', :name, '%')) " +
                "and interlocutor.id <> :id",
        nativeQuery = true
)
List<Interlocutor> findByFilters(
        @Param("state") String state,
        @Param("city") String city,
        @Param("name") String name,
        @Param("id") UUID id
);
```

Figura 9. Assinatura do método de busca de usuários e consulta definida via anotação

Além da disponibilização deste *endpoint*, outros dois *endpoints* foram necessários para a implementação completa destes casos de uso, um para listagem das mensagens; responsável também pela marcação de lida das mensagens buscadas; que será requisitado cada vez que uma mensagem chegar ao tópico destinada à alguma conversa que o usuário esteja. E por fim, um *endpoint* para criação das conversas que unirá as interfaces de usuário de busca de interlocutores com a principal, que conterá as conversas que o usuário participa, as mensagens da conversa selecionada e um campo de texto.

Com a disponibilização destes *endpoints*, foi feito o desenvolvimento da interface de usuário principal do sistema contendo a listagem das conversas que o usuário autenticado participa, sendo possível selecionar uma conversa para visualizar e interagir; as mensagens da conversa selecionada da listagem com data – hora de envio e indicador de mensagem lida ou não; além do componente para entrada de texto responsável pela integração do *front-end* com o *endpoint* de envio de mensagem. Como é possível ver pela Figura 11 onde temos as funcionalidades citadas anteriormente. Além dos *scripts* para cada um dos *endpoints* já citados, disponibilizados neste incremento, para garantir a reatividade desta interface, foi necessário realizar chamadas ao *back-end* a cada mensagem que chega ao *broker* destinada a um *chat* do qual o usuário autenticado participe, para atualizar a listagem de conversas com o intuito de mostrar a quantidade de mensagens não lidas e a última mensagem de forma atualizada. Também

é necessário buscar as mensagens da conversa selecionada, caso a mensagem que acabou de chegar ao *broker* seja destinada a ela.

```java
@Transactional  1 usage  Lucas Carmassi
public void send(SendMessageRequest payload, UUID interlocutorSenderId) {
    var chat = this.chatService.get(payload.chatId());

    Message replyTo = null;

    if (payload.replyToId() != null)
        replyTo = this.messageRepository.findById(payload.replyToId()).orElseThrow(MessageNotFoundException::new);

    var sender = new Interlocutor();
    sender.setId(interlocutorSenderId);

    this.messageRepository.save(new Message(
            payload.content(),
            replyTo,
            sender,
            chat
    ));

    entityManager.flush();

    messagingTemplate.convertAndSend( destination: "/topic/messages", chat.getId());
}
```

Figura 10. Método de cadastro de mensagem e publicação no tópico

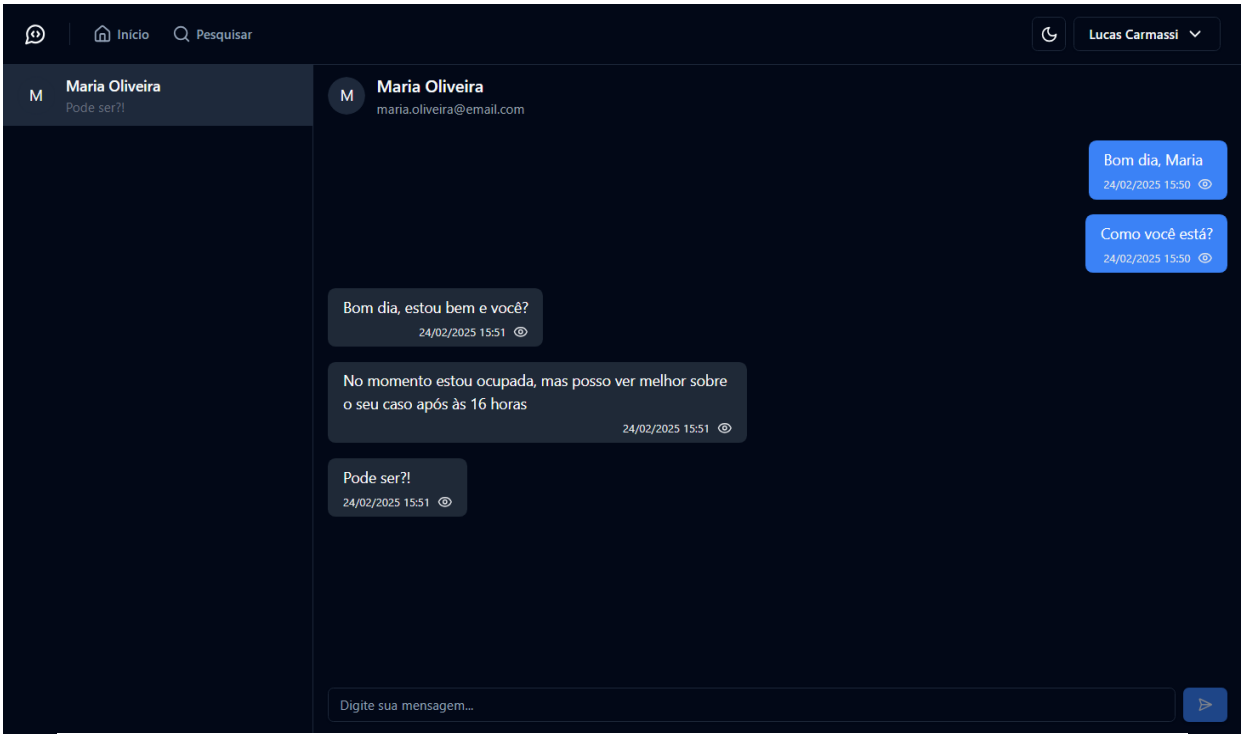


Figura 11. Interface de usuário para listagem de conversas, envio e leitura de mensagens

5.3. Testes Unitários e de Integração

Os testes automatizados realizados durante a implementação dos incrementos consistem

principalmente em testes unitários e de integração. Os testes unitários verificam a menor unidade testável da aplicação de forma isolada e podem cobrir funcionalidades por meio do acesso ao código. Já os testes de integração validam a interação entre as unidades, garantindo seu funcionamento em conjunto. Esse tipo de teste é mais complexo, pois depende de mocks e persistência de dados, além de demandar mais processamento e tempo de execução, resultando em uma quantidade menor de testes em comparação aos unitários (ZUP, 2021).

Na aplicação *back-end*, os testes foram desenvolvidos utilizando o *framework* de automação de testes JUnit, tanto para os testes unitários quanto para os de integração. A Figura 12 apresenta um exemplo de teste unitário para a classe da camada *service*, responsável pela implementação das regras de negócio que envolvem a entidade Interlocutor. Esse teste tem como objetivo garantir que o método de criação de um novo interlocutor esteja funcionando conforme o esperado, validando o caso de sucesso no cadastro de um usuário no sistema. Esse método de teste consiste em três etapas diferentes: primeiro, foram criados dados fictícios responsáveis pela formação dos *mocks* e das validações do cenário de teste. Na segunda etapa, são realizados os *mocks* necessários para a execução do caso de sucesso do cenário, como a atribuição do retorno do método de *encode* de senha como “senhacodificada”. Por fim, a chamada ao método testado é feita, e as validações que garantem que o cenário do teste esteja funcionando corretamente são aplicadas, como a validação da senha criptografada e validação de não nulo do interlocutor salvo.

```
109     @Test  Lucas Carmassi
110     @DisplayName("Deve criar interlocutor com sucesso")
111     void deve_criar_interlocutor() {
112         Interlocutor interlocutor = request.toInterlocutor();
113         interlocutor.setPassword("senhacodificada");
114
115         when(interlocutorRepository.findByEmail(request.email())).thenReturn( t: Optional.empty());
116         when(passwordEncoder.encode(request.password().trim())).thenReturn( t: "senhacodificada");
117         when(interlocutorRepository.save(any(Interlocutor.class))).thenReturn(interlocutor);
118
119         Interlocutor savedInterlocutor = interlocutorService.createInterlocutor(request);
120         assertNotNull(savedInterlocutor);
121         assertEquals( expected: "senhacodificada", savedInterlocutor.getPassword());
122     }
```

Figura 12. Cenário de teste para validação de cadastro de interlocutor

Para a realização dos testes de integração, foi necessário, antes de escrever os cenários de teste, configurar a classe de testes, disponibilizando um *mock* para realizar chamadas HTTP ao *back-end* e registrando um interlocutor na base de dados. Com isso, foi possível iniciar a implementação dos casos de teste. A Figura 13 apresenta o cenário de autenticação do interlocutor. Como o *setup* da classe já havia sido realizado, o método de teste consiste na criação dos dados específicos para esse cenário, que, nesse caso, correspondem a um objeto contendo e-mail e senha para a autenticação do usuário. Em seguida, utiliza-se o *mock* para realizar a chamada HTTP ao *endpoint* “/interlocutor/auth” e validar se a resposta recebida está de acordo com o esperado, verificando o *status code* e a presença de um *token* no corpo da resposta.

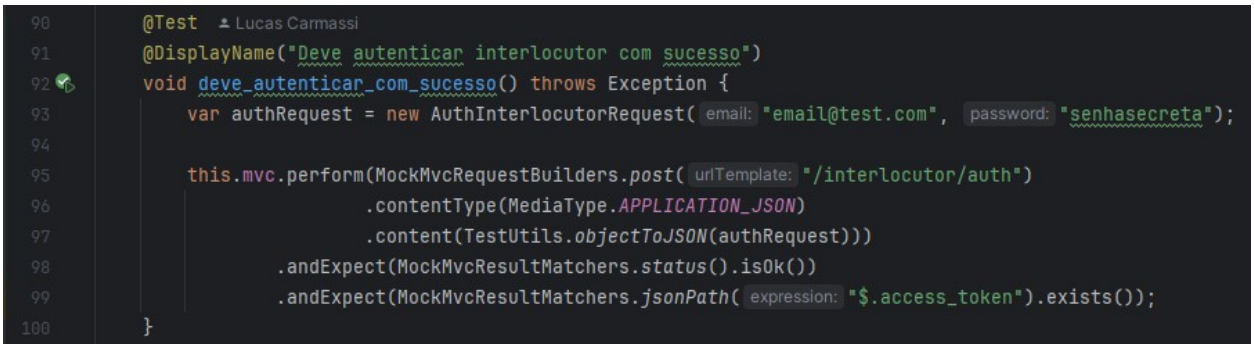


Figura 13. Cenário de teste para validação da autenticação do interlocutor

6. Conclusão

Os estudos necessários para o entendimento acerca do tema dos refugiados trouxeram o compreendimento da importância da troca de experiências e a criação de laços com a comunidade. O propósito deste trabalho é documentar a implementação de um protótipo de sistema *web* para possibilitar o contato de refugiados e imigrantes do Brasil com refugiados e imigrantes que já se estabilizaram em uma nova comunidade, perfil de movimentos sociais que atendem esse público, e moradores da comunidade que estão disposto a ajudar, com intuito de possibilitar a criação de uma rede de apoio para refugiados e imigrantes. Ainda que nem todas as funcionalidades descritas na etapa de Engenharia de Requisitos tenham sido implementadas, o objetivo da criação de um protótipo de sistema de trocas de mensagens de texto foi concluído, disponibilizado em repositórios de código fontes disponíveis por meio dos endereços: https:/github.com/carmassilucas/t2r-backend e https://github.com/carmassilucas/t2r-frontend**.**

Com base na metodologia especificada na Seção 4, o primeiro passo para o desenvolvimento deste sistema foi a análise de três aplicações já existentes no mesmo nicho: sistemas de troca de mensagens de texto em tempo real. Essa análise auxiliou a etapa seguinte da construção do T2R, a fase de Engenharia de Requisitos, na qual foram definidas, com base nessas aplicações, as principais funcionalidades que deveriam estar presentes no protótipo. Nesta etapa, foram elaborados os diagramas de caso de uso e o diagrama de classes. Com isso, foi possível iniciar o desenvolvimento das aplicações de forma incremental, implementando pequenas funcionalidades de maneira iterativa, conforme descrito anteriormente na Seção 5.2.

Neste trabalho foram utilizados conhecimentos adquiridos em algumas disciplinas cursadas ao longo da graduação, como: Linguagem de Programação; Banco de Dados; Engenharia de Software; Análise Orientada a Objetos; Estruturas de Dados; Arquitetura de Software; Programação Orientada a Objetos; Gestão de Projetos; Desenvolvimento de Sistemas Web; e Segurança da Informação. Além disso, para a implementação do protótipo foram necessários conhecimentos sobre tecnologias e ferramentas não aprendidas na graduação, como: o Ecossistema Spring; a plataforma Node JS; e as ferramentas Vite e TailwindCSS.

Com isso, desafios técnicos e estruturais que exigiram adaptações constantes foram enfrentados no processo de desenvolvimento do sistema. A integração com o *front-end* exigiram aprendizados além do conteúdo abordado na graduação, dificuldades na estilização com TailwindCSS e na organização dos formulários exigindo ajustes de *layout*. A implementação

incremental também trouxe desafios na compatibilização entre *back-end* e *front-end*, exigindo refinamento contínuo da arquitetura. Esses obstáculos foram superados com pesquisa e experimentação; e a aplicação prática dos conceitos e conhecimentos adquiridos.

6.1 Trabalhos Futuros

Como o Trabalho de Conclusão de Curso prevê 80 horas de dedicação, algumas funcionalidades identificadas não foram totalmente implementadas. Dentre os trabalhos pendentes, destacam-se as funcionalidades relacionadas à moderação da plataforma, incluindo a criação dos *endpoints* necessários para a funcionalidade de denúncia de mensagens e o desenvolvimento de um projeto *front-end* para moderação, com interfaces para autenticação do moderador, listagem de denúncias, histórico de denúncias analisadas e banimento de interlocutores. Além disso, seria necessária a utilização de bibliotecas de tradução para a plataforma e para as mensagens enviadas pelos usuários, bem como uma solução para a configuração dessas traduções.

Referências

ACNUR. (2024). Alto Comissariado das Nações Unidas para Refugiados. Disponível em: https://www.acnur.org/portugues/. Acesso em: 19 jul. 2024.

AMAZON WEB SERVICES. O que é conteinerização? Disponível em: https://aws.amazon.com/pt/what-is/containerization/. 2023. Acesso em: 5 ago. 2024.

AMAZON WEB SERVICES. Qual é a diferença entre um modelo de dados lógico e um modelo de dados físico? 2024. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-logical-and-physical-data-model/. Acesso em: 21 dez. 2024.

ARRUDA, Filipe Jessé de Castro; MARTINS, Daves Márcio Silva. Análise comparativa entre sistemas de mensageria: Apache Kafka vs RabbitMQ. Revista de Sistemas de Informação, v. 1, n. 1, p. 1-10, 2023. Disponível em: http://periodicos.jf.ifsudestemg.edu.br/revistabsi/article/view/536/150. Acesso em: 09 out. 2024.

BRASIL. Ministério do Esporte. Lei Geral de Proteção de Dados (LGPD). Disponível em: https://www.gov.br/esporte/pt-br/acesso-a-informacao/lgpd. Acesso em: 05 mar. 2025.

MINISTÉRIO DA JUSTIÇA E SEGURANÇA PÚBLICA. Divulgados novos números e perfis de refugiados no Brasil. Disponível em: https://www.gov.br/mj/pt-br/assuntos/noticias/divulgados-novos-numeros-e-perfis-de-refugiados-no-brasil. Acesso em: 17 fev. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. What is PostgreSQL? PostgreSQL, 2023. Disponível em: https://www.postgresql.org/about/. Acesso em: 21 dez. 2024.

RAMOS, N. Diversidade cultural, educação e comunicação intercultural – políticas e estratégias de promoção do diálogo intercultural. Educação em Questão, [S. l.], v. 34, n. 20, p. 9-32, 2009. Disponível em: https://periodicos.ufrn.br/educacaoemquestao/article/view/3941/3208. Acesso em: 13 fev. 2025.

RED HAT. O que é uma API REST? Disponível em: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api. Acesso em: 06 ago. 2024. 2024a

RED HAT. O que é conteinerização? Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-containerization#benef%C3%ADcios. Acesso em: 5 ago. 2024. 2024b

SIGNAL FOUNDATION. Página inicial. Signal. Disponível em: https://signal.org/pt_BR/. Acesso em: 06 ago. 2024.

SOMMERVILLE, I. Engenharia de Software. 10. ed. São Paulo: Pearson, 2015.

TEIXEIRA, ANA CHRISTINA CELANO, SILVA, ELIANA CRISTINA MOTTA DA ; BALOG, DANIELA LONGOBUCCO TEIXEIRA. Por que é tão difícil pertencer? As dificuldades dos refugiados em seus processos de inserção no mercado de trabalho e na sociedade brasileira. Cadernos EBAPE.BR, v. 19, n. 2, p. 265–277, 2021. Disponível em: https://doi.org/10.1590/1679-395120200016.

VIBER. Página inicial. Viber. Disponível em: https://www.viber.com/pt-pt/. Acesso em: 06 ago. 2024.

ZUP. Tipos de teste: Guia completo. Disponível em: https://zup.com.br/blog/tipos-de-teste#texto-blog. Acesso em: 04 mar. 2025.

Apêndice I

LEVANTAMENTO DE REQUISITOS DO SISTEMA TALK TO REFUGEE

REQUISITOS FUNCIONAIS:

1. **Autenticação do usuário**
    a. Os interlocutores devem poder acessar a plataforma com suas credenciais.
    b. Os interlocutores devem poder se cadastrar na plataforma
    c. Os interlocutores devem poder se desconectar da plataforma
2. **Gerenciamento de perfil de interlocutor**
    a. Os interlocutores devem poder visualizar suas informações
    b. Os interlocutores devem poder alterar suas informações
    c. Os interlocutores devem poder alterar sua senha
    d. Os interlocutores devem poder alterar sua foto
    e. Os interlocutores devem poder desativar sua conta
3. **Busca de interlocutores**
    a. Os interlocutores devem poder buscar outros interlocutores
        i. A busca deve poder ser feita por localização
        ii. A busca deve poder ser feita por tipo de perfil
        iii. A busca deve poder ser feita por nome
    b. Os interlocutores devem poder iniciar uma conversa pelo resultado da busca de interlocutores
    c. Os interlocutores devem poder criar um grupo pelo resultado da busca de interlocutores
4. **Conversas privadas**
    a. Os interlocutores devem poder listar as conversas já existentes
    b. Os interlocutores devem poder visualizar mensagens já enviadas
    c. Os interlocutores devem poder enviar novas mensagens
        i. A mensagem pode ser uma resposta a outra mensagem
    d. Os interlocutores devem poder enviar arquivos
    e. Os interlocutores devem poder denunciar uma mensagem
5. **Conversas em grupo**
    a. Os interlocutores devem poder listar os grupos em que estão
    b. Os interlocutores devem poder enviar mensagens na conversa em grupo
        i. A mensagem pode ser uma resposta a outra mensagem
    c. Os interlocutores devem poder alterar as informações do grupo
    d. Os interlocutores devem poder alterar a foto da conversa em grupo
    e. Os interlocutores devem poder sair do grupo
    f. Os interlocutores devem poder denunciar uma mensagem
6. **Moderação da plataforma**
    a. O administrador deve poder acessar o sistema com credenciais

previamente cadastradas

b. O administrador deve poder listar perfis denunciados
    i. As mensagens denunciadas deste perfil serão listadas
c. O administrador deve poder desativar a conta de um interlocutor
d. O administrador deve poder se desconectar da plataforma

REQUISITOS NÃO FUNCIONAIS:

1. **Desempenho**
    a. O sistema deve ter um tempo máxima de 1s para resposta de qualquer requisição
    b. O sistema deve ser escalável a fim de suportar o crescimento no número de usuários e de dados
2. **Segurança**
    a. O sistema deve garantir a proteção dos dados do usuário
        i. Não expor dados sigilosos dos usuários
    b. O sistema deve ter políticas de autenticação seguras
    c. O sistema deve estar de acordo com a Lei Geral de Proteção de Dados
        i. Não solicitar dados sensíveis que não serão utilizados para agregar valor ao usuário
3. **Portabilidade**
    a. O sistema deve ser compatível com diferentes especificações
        i. O sistema deve ser compatível com diferentes *browsers*
        ii. O sistema deve ser compatível com diferentes *SOs*
        iii. O sistema deve ser compatível com diferentes dispositivos
4. **Observabilidade**
    a. Ferramentas de monitoração e logging devem ser implementadas
        i. Deve ser possível ter acesso aos logs para acompanhamento do fluxo de execução e análises de erros
        ii. Deve ser possível ter acesso à informações métricas para acompanhar o desempenho da aplicação
5. **Manutenibilidade**
    a. O código deve ser modular, facilitando adição de novas funcionalidade, refatoração e manutenção das existentes
    b. O código deve ter um mínimo de 75% de cobertura de testes
        i. Testes unitários devem garantir o funcionamento correto das unidades
        ii. Testes de integração devem garantir o funcionamento dos fluxos da aplicação

Documento Digitalizado Público

Anexo I - Artigo final

Assunto: Anexo I - Artigo final
Assinado por: Daniela Marques
Tipo do Documento: Projeto
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Daniela Marques**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 05/03/2025 20:38:52.

Este documento foi armazenado no SUAP em 05/03/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1956474
Código de Autenticação: f93367d27c