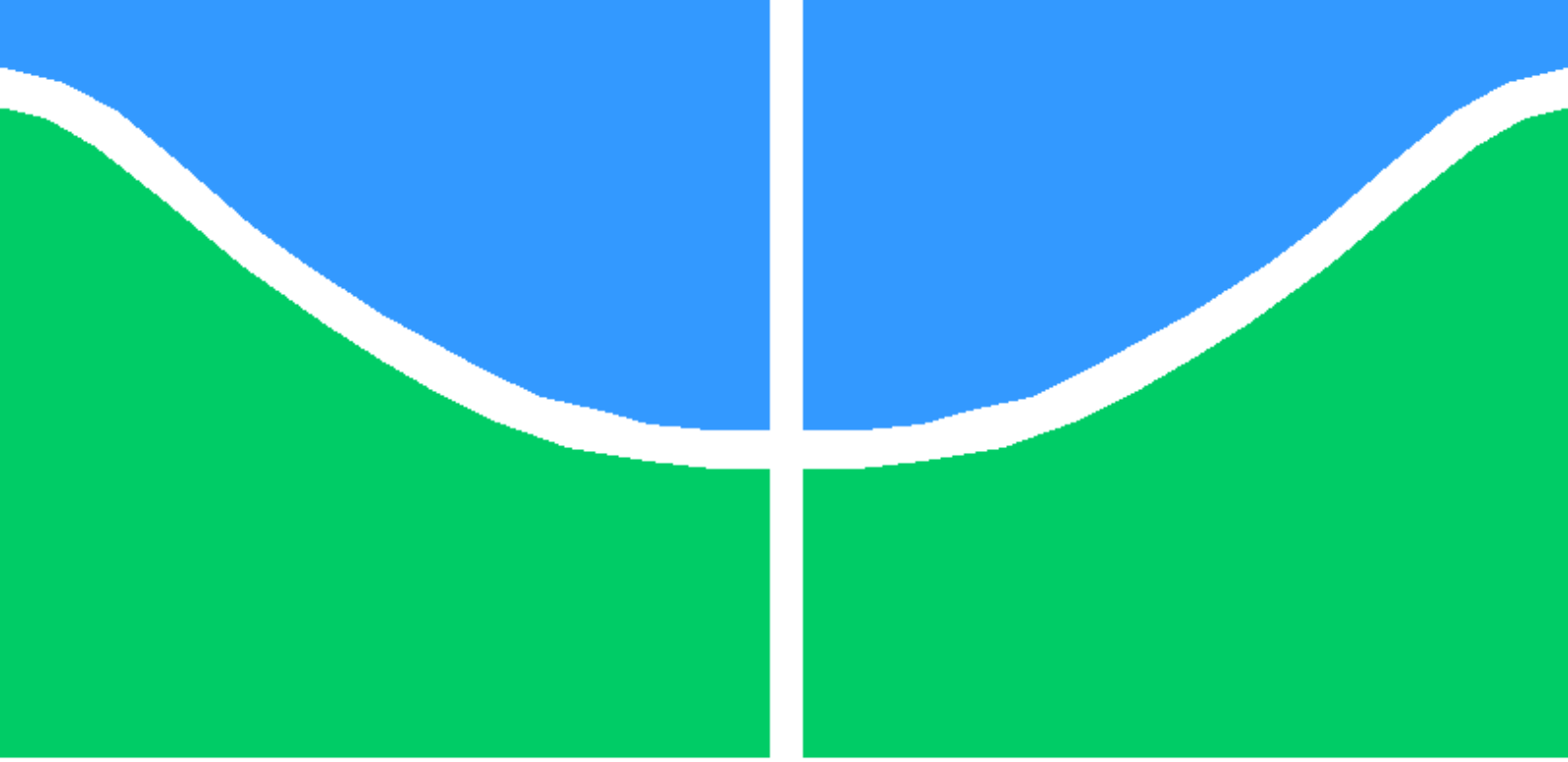

Trabalho Final de Graduação

Estudo de aplicação de segurança colaborativa baseada em geolocalização

Eric Serka do Carmo Rodrigues

Brasília, Fevereiro de 2023

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Trabalho Final de Graduação

Estudo de aplicação de segurança colaborativa baseada em geolocalização

Eric Serka do Carmo Rodrigues

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. PhD. Ricardo Staciarini Puttini, _____
ENE/UnB
Orientador

Prof. PhD. Rafael Timóteo de Sousa Júnior, _____
ENE/UnB
Examinador

Prof. PhD Fabio Lucio Lopes de Mendonça, _____
ENE/UnB
Examinador

Agradecimentos

Agradeço a minha família, aos meus amigos, aos meus professores e aos meus orientadores.

Este trabalho contou com suporte do Laboratório de Tecnologias da Tomada de Decisão da Universidade de Brasília (LATITUDE/UnB), que tem apoio do CNPq - Conselho Nacional de Pesquisa (Outorgas 312180/2019-5 PQ-2 e 465741/2014-2 INCT em Cibersegurança) e da Universidade de Brasília (Outorga FUB/COPEI 7129).

Eric Serka do Carmo Rodrigues

Resumo

A Constituição Federal de 1998 prevê como dever do Estado garantir segurança pública aos seus cidadãos. No entanto, o Brasil sempre esteve entre os países mais violentos nos últimos anos, de acordo com a Organização Mundial da Saúde (OMS).

Casos de roubos, sequestros e latrocínios são frequentemente vistos nos jornais. Além disso, com a sobrecarga de trabalho e o baixo salário, o crime organizado passou a ser praticado também pelas corporações policiais. Tudo isso personifica a deficiência da segurança do país e aumenta a sensação de insegurança e medo por parte da população.

Com a era dos *smartphones* (celulares inteligentes), cada vez mais aplicativos vêm sendo desenvolvidos a fim de resolver problemas cotidianos das pessoas. Não é diferente se tratando do quesito segurança. Esse trabalho contém um estudo dos componentes que compõe uma aplicação desse gênero. Estudo esse que foi desde a identificação do problema, desenvolvimento da aplicação proposta, exposição e análise dos resultados até a sua conclusão e sugestões de trabalhos futuros.

Palavras-chaves: Segurança pessoal, segurança colaborativa, aplicativos móveis, aplicações distribuídas, aplicações multiusuário.

Abstract

The Federal Constitution of 1998 provides for the State's duty to guarantee public security to its citizens. However, Brazil has always been among the most violent countries in recent years, according to the World Health Organization (WHO).

Cases of kidnapping and robberies are frequently seen in the news. In addition, with work overload and low wages, organized crime was also practiced by police corporations. All this personifies the country's security deficiency and increases the population's sense of insecurity and fear.

With the era of smartphones, more and more applications have been developed in order to solve people's everyday problems. It is no different when it comes to security. This work contains a study of the components that make up an application of this kind. This study ranged from identifying the problem, developing the proposed application, exposing and analyzing the results to its conclusion and suggestions for future work.

Keywords: Personal security, collaborative security, mobile applications, distributed applications, multiuser applications.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Acrônimos

UnB	Universidade de Brasília
API	Application Programming Interface
OMS	Organização Mundial da Saúde
WHO	World Health Organization
FE	Faculdade de Educação
RFC	Request for Comments
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
CSWH	Cross-Site Websocket Hijacking
JSON	JavaScript Object Notation
JWT	JSON Web Token
XSS	Cross-site scripting
SQL	Structured Query Language
ws	WebSocket
wss	WebSocket Secure
TLS	Transport Layer Security
HTTPS	Hyper Text Transfer Protocol Secure
CSRF	Cross-Site Request Forgery
APP	Application
SMS	Short Message Service
GPS	Global Positioning System
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
URL	Uniform Resource Locator
PWA	Progressive Web App
ORM	Object-Relational Mapping
OTP	One-Time Password
UI	User Interface
LGPD	Lei Geral de Proteção de Dados Pessoais
MVP	Minimum Viable Product

1 Introdução

Aplicativos móveis são programas computacionais desenvolvidos para serem usados em dispositivos eletrônicos móveis como relógios inteligentes, *smartphones*, *tablets* etc. Aplicativos deste tipo estão em forte crescimento principalmente por conta da grande quantidade desses dispositivos eletrônicos e seus usuários ativos. Tais dispositivos vem se tornando cada vez mais populares por possuírem interface amigável e fácil usabilidade, podendo ser utilizados por crianças e até idosos. Por esses e outros motivos, aplicativos móveis possuem papeis cada vez mais importantes na vida das pessoas e tem chamado a atenção de empresas e investidores.

Uma característica da maioria dos aplicativos móveis é que eles funcionam sob aplicações multiusuários. Por aplicativo móvel entende-se o aplicativo instalado no *smartphone*, enquanto que por aplicação entende-se o sistema como um todo que torna possível a utilização do aplicativo. Aplicações multiusuários são aquelas que suportam dois ou mais usuários simultâneos. Como exemplos de aplicativos que funcionam sob aplicações multiusuários, tem-se aqueles do ramo do *e-commerce* (comércio eletrônico), redes sociais e jogos online. Com o mundo cada vez mais conectado, a demanda por aplicações de usuário único tem diminuído consideravelmente: os *softwares* estão se tornando cada vez mais distribuídos.

Softwares e a tecnologia num geral também têm papel importante em ambientes colaborativos ou situações de colaboração. Por exemplo, atualmente existem diversas ferramentas úteis para o trabalho remoto ou para a educação a distância como por exemplo *Slack* [40], *Trello* [19], *GitHub* [20], *Microsoft Teams* [64] e *Google Meet* [22]. A respeito de situações de colaboração, existem várias, mas o foco aqui será dado àquelas baseadas em geolocalização, que é um recurso que permite identificar a posição geográfica de objetos e pessoas com base em coordenadas. Exemplos de aplicativos baseados em geolocalização são: *Google Maps* [21], *Waze* [26], *Colab* [57] e *Life360* [23]. Os dois primeiros são bem populares e são utilizados para instruções de direção e atualizações do tráfego em tempo real. Já o terceiro se trata de um aplicativo voltado para cidadãos e governos. O objetivo dele é colaborar com a construção de uma sociedade melhor por meio da participação ativa dos cidadãos na formação de políticas públicas e nas decisões da prefeitura. É possível comunicar ocorrências e melhorias a respeito do meio ambiente, água, esgoto, trânsito etc. Por último, tem-se o *Life360*, aplicativo de rastreamento de pessoas membros de um grupo comum em tempo real.

Esse trabalho propõe um estudo dos componentes envolvidos na criação de uma aplicação multiusuário de segurança colaborativa baseada em geolocalização. Nesse estudo, será demonstrada a concepção e implementação simplificada da aplicação, considerando as seguintes funcionalidades básicas:

1. permitir a criação de grupos de usuários que confiam e colaboram entre si, compartilhando dados pessoais, inclusive sua localização, em tempo real;

2. realizar rastreamento, em tempo real, da localização de pessoas que fazem parte de um mesmo grupo;
3. disponibilizar chats de conversa entre membros de grupos;
4. permitir a criação de notificações instantâneas para solicitar ajuda para todas as pessoas de grupo (botão de pânico);
5. armazenar localidades (coordenadas geográficas) classificadas como possivelmente perigosas por membros do grupo e mostrá-las no mapa de navegação.

1.1 Objetivos

O presente trabalho apresenta os objetivos gerais e específicos descritos a seguir.

1.1.1 Objetivo Geral

Estudar os componentes necessários de uma aplicação distribuída e multiusuário, baseada em geolocalização e com foco na segurança colaborativa de um ou mais grupos fechados. A aplicação possibilita, principalmente, a criação de grupos de usuários que compartilham dados pessoais entre si, em tempo real, e utiliza esses dados para disponibilizar o rastreamento de localização, em tempo real, e emissão de alertas para casos de emergência entre membros de um mesmo grupo.

1.1.2 Objetivos específicos

- Compreender os componentes que fazem parte da aplicação a ser desenvolvida, que são eles: *frontend* (aplicativo móvel), *backend* (servidor) e banco de dados;
- Estudar sobre *websockets* e como eles auxiliam uma aplicação a ter funcionalidades que respondem em tempo real;
- Estudar sobre plataformas e *frameworks* para desenvolvimento de aplicativos móveis;
- Estudar sobre tipos de interação entre aplicações e bancos de dados;
- Descrever a aplicação a ser desenvolvida funcionalmente;

1.2 Justificativa

Como já mencionado anteriormente, o Brasil está na lista dos países mais violentos do mundo, segundo a OMS [41], há um bom tempo. Tanta exposição à violência e à falta de segurança resultam em prejuízos sociais, como atraso no crescimento econômico e impactos na saúde pública, e individuais, como efeitos deletérios físicos e mentais para as vítimas. Quem passa por situações traumáticas corre o risco de desenvolver uma série de reações que, com o tempo, podem se configurar em transtornos mentais [43].

O cenário na própria UnB não é diferente. Recentemente, no ano de 2022, ocorreram episódios de assédio e violência sexual sofridos por alunos e alunas nas dependências do campus Darcy Ribeiro – Asa Norte. Atos como estes praticados são um ataque contra a dignidade das vítimas mas não atingem apenas a elas, representam também um ataque a todos e todas. A própria Faculdade de Educação (FE) solicitou, em uma nota de repúdio publicada pela mesma contra os episódios, uma ampliação e intensificação das ações de segurança (como aquelas voltadas à proteção em rede) além de uma melhor vigia, monitoramento e iluminação nos pontos do campus [38].

A aplicação proposta por esse trabalho é própria para uso de pessoas pertencentes a um mesmo círculo social, como entes queridos ou amigos, e exige confiança mútua. Ela pode promover um aumento na segurança de seus usuários por conta do monitoramento constante de localizações por pessoas de confiança e sempre levando em conta a privacidade. Além disso, os usuários colaborariam fornecendo localidades classificadas como possivelmente perigosas, possibilitando assim que outros usuários as evitem.

1.3 Organização do trabalho

O trabalho possui cinco capítulos, sendo este o primeiro. O capítulo 2 trata de uma revisão teórica acerca de conceitos que foram importantes para a concepção do trabalho como um todo além de uma revisão de pesquisas e trabalhos anteriores com tema relacionado ao do presente trabalho. O capítulo 3 tem como objetivo mapear todas as funcionalidades da aplicação final desenvolvida. Já o capítulo 4 faz um melhor detalhamento acerca da implementação da aplicação abordando assuntos como tecnologias, ferramentas, arquitetura e resultados. Por fim, o capítulo 5 apresenta ideias de melhorias que podem ser aplicadas em trabalhos futuros além da conclusão do estudo realizado no presente trabalho.

2 Fundamentos, Tecnologias e Revisão Bibliográfica

2.1 Fundamentos e Tecnologias

Nesta seção, serão apresentados conceitos importantes para a contextualização do trabalho, com o objetivo de fornecer base teórica para os temas relacionados ao desenvolvimento do mesmo.

Mais detalhes quanto às tecnologias que foram escolhidas e utilizadas no desenvolvimento da aplicação proposta se encontram no capítulo 5.

2.1.1 *WebSockets*

O protocolo HTTP [62], que teve sua versão 1.1 definida no RFC 2616 [51], é capaz de fornecer uma comunicação pela Internet conveniente com troca de informações, hiperlinks e sem estado, mas pode ser problemático quando a troca de dados em tempo real é necessária. Implementar uma comunicação em tempo real (ou algo perto disso) utilizando de HTTP não é impossível, mas o preço é muito alto: latência aumentada e alto tráfego de rede [44].

Para aplicações que dependem de eventos acontecendo em tempo real e a **escalabilidade** é desejável, o *WebSocket* (ws) [66] é recomendado. Ele é um protocolo definido no RFC 6455 [52] capaz de fornecer comunicação eficiente, com baixa latência, baixa sobrecarga da rede e com estado entre servidores e clientes da web [53].

Enquanto o HTTP opera no modo *half-duplex* para transmitir seus dados (sentido de transmissão bidirecional mas não simultâneo [34]), o *WebSocket* permite a comunicação *full-duplex* (sentido de transmissão bidirecional e simultâneo [34]) em um único socket TCP [44].

O protocolo *WebSocket* consiste em duas partes. A primeira é o *handshake*, que é quem estabelece e encerra a conexão. E a segunda são os próprios dados, mensagens enviadas entre cliente e servidor.

O remetente fornece uma mensagem ou binário como carga e o receptor é notificado da sua entrega quando toda a mensagem está disponível. Para tornar isso possível, o *WebSocket* usa um formato de quadro (*frame*) personalizado que divide cada mensagem do aplicativo em um ou mais quadros, transporta para o destino, remonta e, finalmente, notifica o destinatário quando a mensagem inteira é recebida [53].

Para o *handshake*, o cliente faz um HTTP GET juntamente com um *upgrade* que solicita para o servidor que o protocolo seja alterado para o *WebSocket*. Na segunda parte, o servidor responde com um HTTP 101 (*Switching Protocols* [1]), aceitando a mudança de protocolo [53].

A figura 2.1, extraída de [67], ilustra o funcionamento dos *WebSockets*, da conexão à comunicação.

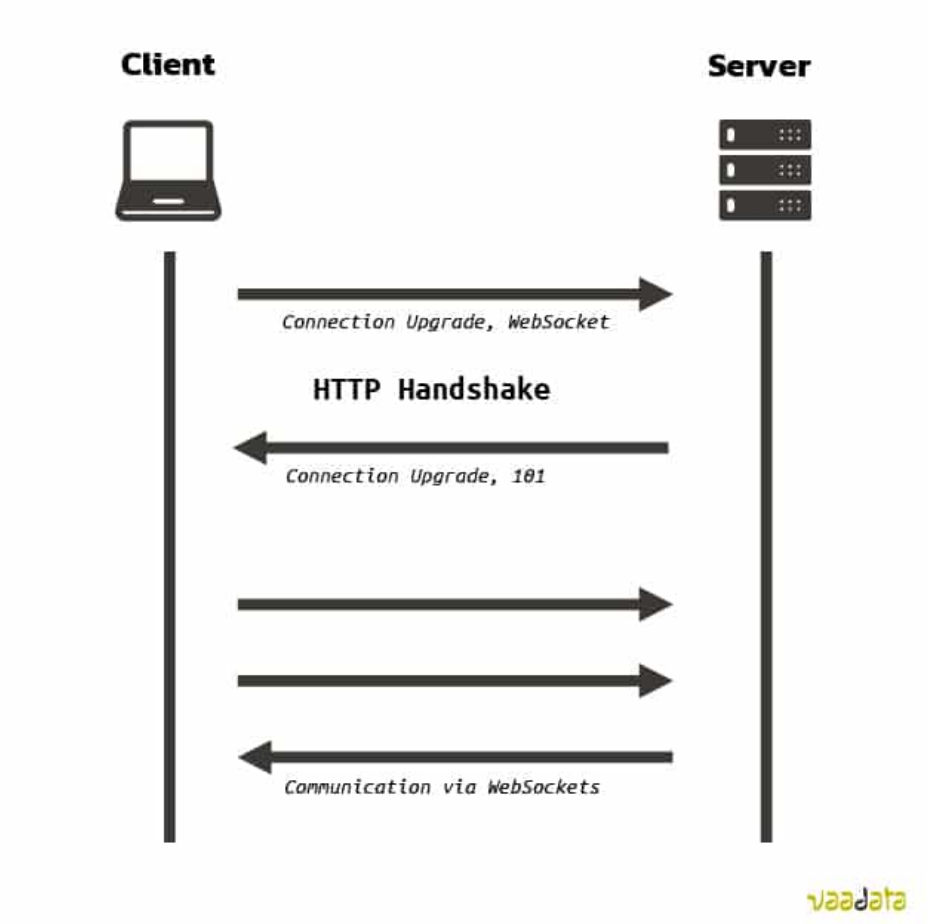

Figura 2.1 – Funcionamento dos *WebSockets*. Fonte: [67]

O que ocorre no HTTP é que o cliente inicia uma solicitação e aguarda uma resposta. Isso é chamado de transação. Ou seja, cada solicitação/resposta (*request/response*) no protocolo HTTP inicia uma transação diferente. Já o *WebSocket* inicia uma transação com ciclo de vida longo (*long-life cycle*). Isso significa que em uma mesma transação no protocolo *WebSocket* podem ocorrer vários *requests* e *responses* diferentes. Além disso, com o *WebSocket*, o servidor pode enviar dados para o cliente mesmo que nenhuma solicitação tenha sido feita por ele, desde que o processo de *handshake* já tenha sido realizado [67].

Apesar de algumas vantagens em relação ao protocolo HTTP para aplicações em tempo real, o *WebSocket* possui as vulnerabilidades habituais do HTTP, algumas com algumas especificidades [67]. Dessa forma, independente do protocolo utilizado na aplicação, é papel do arquiteto/desenvolvedor de *software* limitar os riscos se utilizando do ferramental necessário. Abaixo se encontra uma lista [67] com possíveis vulnerabilidades e ataques contra *WebSockets*.

- **Controle de autenticação:** *WebSockets* não possuem um mecanismo nativo de autenticação. Portanto, uma solução limpa como *cookies* ou autenticação JWT deve ser implementada.
- **Controle de autorização:** Assim como no caso da autenticação, *WebSockets* não possuem um sistema nativo de gerenciamento de autorizações em que os usuários podem acessar apenas

os dados e serviços aos quais devem ter acesso. Isso permite que um invasor expanda seus privilégios verticalmente ou acesse os dados de um outro usuário com os mesmos privilégios que ele [67].

- **Riscos relacionados à entrada do usuário:** Dados inseridos pelos usuários são a maior causa de ataques. Por conta disso, todas as entradas devem ser tratadas com o método mais adequado de acordo com o contexto antes de serem utilizadas. Alguns dos ataques de entrada mais comuns são: XSS, Injeção de SQL e injeção de código [67].

- **Riscos de *sniffing*:** A transmissão de dados no *WebSocket* é feita em texto não criptografado, assim como no HTTP. Portanto, é possível recuperar esses dados por meio de ataques *man-in-the-middle* (homem no meio). Para evitar o vazamento de dados, o protocolo *WebSocket Secure* (wss) deve ser implementado. Importante salientar que, assim como no HTTPS, wss não significa que a aplicação seja segura. É apenas o transporte criptografado de dados via TLS.

- ***Cross-Site Websocket Hijacking* (CSWH):** É um ataque semelhante ao CSRF [8]. Ele é possível quando o servidor depende apenas dos dados de autenticação da sessão (*cookies*) para realizar uma ação autenticada. Sabe-se que o canal de comunicação entre cliente e servidor é criado de acordo com a origem do pedido de abertura. Assim, se a solicitação de alteração de protocolo (*handshake*) for baseada apenas em *cookies*, um invasor pode interceptar uma vítima para iniciar uma solicitação com sua sessão, mas no servidor dele (e não no servidor da aplicação real). Uma vez feito o ataque, o invasor pode se comunicar com o servidor via *WebSockets* sem o conhecimento da vítima. O invasor pode, portanto, executar ações no lugar de um usuário e também ler as mensagens enviadas pelo servidor via *WebSockets*. Para solucionar esta vulnerabilidade, é necessário adicionar um *token* único por sessão e que não possa ser adivinhado como parâmetro da solicitação de *handshake* [67].

2.1.2 Plataformas e frameworks para desenvolvimento de aplicativos móveis

As principais plataformas de dispositivos móveis são o Android [3] e o iOS [27]. Elas possuem grandes diferenças entre si, como as linguagens de programação utilizadas e os ambientes de desenvolvimento.

Nas próximas seções, serão discutidas três abordagens diferentes para o desenvolvimento de aplicativos móveis.

2.1.2.1 Desenvolvimento Nativo

Aplicativos nativos são aqueles desenvolvidos especificamente para uma plataforma (Android ou iOS) que permitem o uso pleno dos recursos dos dispositivos. Eles são escritos em linguagens de programação de alto nível como Java [28] ou Kotlin [32], para Android, e Objective-C [2] ou Swift [69], para iOS [47].

Por muito tempo, o desenvolvimento nativo de aplicativos móveis foi a abordagem mais comum. O maior inconveniente dessa abordagem é a necessidade de se ter dois projetos completamente diferentes desenvolvidos, caso o aplicativo ser multiplataforma seja desejável.

O desenvolvimento nativo geralmente é feito quando os requisitos de desempenho e usabilidade são altos, há necessidade de usar uma grande quantidade de animações (como acontece em jogos), é necessário acessar APIs específicas do dispositivo ou quando, por algum motivo, o aplicativo é destinado e exclusivo para apenas usuários de determinada plataforma. As APIs nativas são fornecidas ao desenvolvedor por meio do SDK da plataforma. É através dessas APIs que o aplicativo se comunica com o *hardware* do dispositivo móvel como por exemplo câmera, *bluetooth*, GPS, acelerômetro, bússola, que normalmente não podem ser acessadas diretamente por um aplicativo web executando em um navegador de internet [47].

Aplicativos nativos tem o visual parecido com o próprio dispositivo porque os componentes visuais como por exemplo botões, menus e listas são fornecidos e estilizados pela própria plataforma.

2.1.2.2 Desenvolvimento de *web apps*

Web apps são aplicativos para celulares desenvolvidos com tecnologias *web* como JavaScript [29], CSS [9] e HTML [24] mas aperfeiçoados para funcionar como um aplicativo móvel [47].

Por serem executados no navegador, *web apps* possuem como vantagem serem compatíveis com todas as plataformas, serem de fácil atualização e serem mais leves. Já que o aplicativo não é instalado no dispositivo, muita memória de armazenamento e processamento é economizada [47]. A principal desvantagem é o limitado acesso aos recursos de *hardware* do dispositivo e a necessidade de conexão com a internet para funcionamento completo do aplicativo. Ainda pode ser possível visualizar o aplicativo offline por conta do cache do navegador mas o restante da aplicação em si pode se tornar não funcional.

Uma alternativa mais elaborada/avançada e interessante para *web apps* tradicionais são os PWA. Resumidamente, com um PWA, é possível usar mais recursos do dispositivo do que com um *web app* tradicional e, além disso, ao contrário de um *web app* tradicional, a instalação de um PWA em um dispositivo móvel é totalmente (e facilmente) possível [65].

2.1.2.3 Desenvolvimento Híbrido

Aplicativos híbridos são aqueles em parte nativos e em parte *web apps*. Eles tem como finalidade funcionar em diversos dispositivos e plataformas diferentes utilizando o mesmo código fonte. Assim como os aplicativos nativos, os híbridos são instalados a partir da loja de aplicativos da plataforma, podem aproveitar todo o hardware do dispositivo e são executados em ambiente de processo nativo. Existe uma espécie de ponte que conecta o código não nativo aos recursos nativos do dispositivo [47].

Semelhante aos *web apps* tradicionais, precisam de conexão com a internet para funcionamento completo. Eles até podem operar offline mas não fornecerão aos usuários a experiência completa [36], o que não acontece nos aplicativos nativos, já que a conexão com a internet nesse caso não é

realmente necessária [47].

Eles são desenvolvidos usando tecnologia *Web* mas baseados em padrões nativos. Por esse motivo, diz-se que aplicativos híbridos são uma categoria especial de *web apps* [47]. Eles agrupam o conteúdo HTML em tela cheia e não exibem a barra de endereço ou outros controles do navegador.

As principais e atuais tecnologias do mercado para desenvolvimento híbrido de aplicativos móveis são React Native [49], que utiliza a linguagem de programação JavaScript [29], e Flutter [18], que utiliza a linguagem de programação Dart [10].

As principais vantagens desse tipo de desenvolvimento são [47]:

- redução da **complexidade**;
- redução de código;
- redução do tempo de desenvolvimento;
- fácil manutenção;
- diminuição de conhecimento necessário sobre a API;
- aumento de participação no mercado.

2.1.3 ORM, *Query Builder* e *Raw SQL*

Bancos de dados podem ser fundamentais para uma aplicação. Os dois principais tipos são: bancos SQL e bancos noSQL. O primeiro é chamado de banco relacional, se baseia no fato de que os dados sejam armazenados em tabelas. Já o segundo é chamado de não relacional, foi criado a fim de ter uma performance melhor, ser mais escalável em comparação com os tradicionais relacionais e os dados podem ser organizados de diversas maneiras como documentos, colunas, grafos, chave-valor etc. [59].

Nesse trabalho foram analisados três mecanismos de interação entre uma aplicação e um banco de dados relacional: *Raw SQL* (ou *Plain SQL*), *Query Builder* e ORM. Tais mecanismos exigem uma grande troca entre produtividade e controle [72]. A figura 2.2, extraída de [72], mostra esse conceito graficamente. Nela, é possível observar que:

- com o *Plain SQL*, o desenvolvedor tem total controle sobre as operações de banco de dados realizadas porém com o custo de uma baixa produtividade;
- com o *SQL Query Builder*, o desenvolvedor tem um alto controle sobre as operações de banco de dados e uma média produtividade;
- com o ORM, o desenvolvedor perde bastante controle sobre as operações de banco de dados mas ganha bastante no quesito de produtividade.

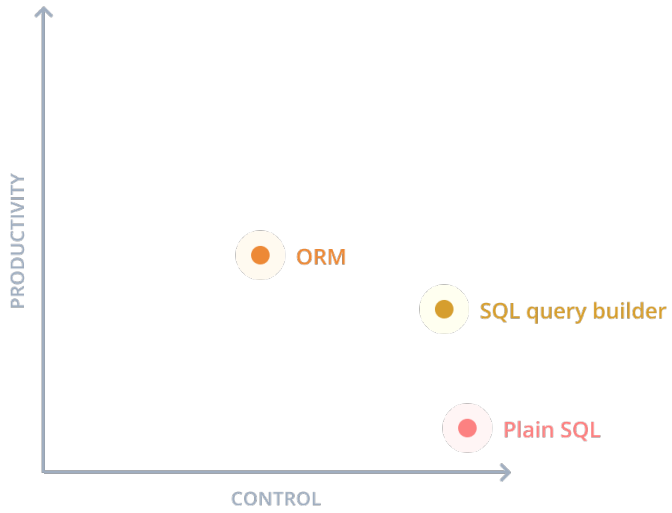

Figura 2.2 – Comparação de ferramentas de banco de dados. Fonte: [72]

A seguir, serão comentadas com mais detalhes as ferramentas mencionadas. Serão evidenciadas as particularidades, vantagens e desvantagens de cada uma. É esperado que a interpretação das informações apresentadas comprovem o conteúdo da figura 2.2.

2.1.3.1 *Raw SQL*

Raw significa "cru" em português. Ou seja, *Raw SQL* é o SQL puro, nu e cru. Também conhecida como *Plain SQL*, é a maneira mais básica e de mais baixo nível de comunicação: é dito ao banco de dados o que fazer explicitamente na sua própria linguagem (SQL).

Esse tipo de comunicação exige que o desenvolvedor tenha um maior conhecimento em SQL, já que não há uma camada superior realizando abstrações. A vantagem disso é que é possível escrever *queries* (consultas ao banco) muito mais performáticas, de maneira flexível e livres de "mágica". Com *Raw SQL*, é possível entender o que cada linha de código faz de maneira muito simples. Porém, dependendo do nível de complexidade da sua aplicação, escrever *queries* SQL na mão pode se tornar uma tarefa bastante complicada.

Esse foi o tipo pioneiro de comunicação, utilizado nas primeiras aplicações conhecidas. Um dos maiores (e mais famosos) problemas dessa abordagem é o chamado *SQL Injection* (injeção SQL), em que um usuário mal intencionado (*hacker*) consegue "injetar" código malicioso em instruções SQL via *inputs* de página WEB ou mobile [58]. Um outro problema que não envolve segurança mas pode afetar a produtividade do desenvolvedor é a falta de suporte dos editores de texto. Como uma *query* é interpretada como uma *string* no editor, geralmente ele não é capaz de detectar erros de digitação, realce de sintaxe ou completar automaticamente códigos SQL [42].

2.1.3.2 *Query Builder*

Query Builders, como o nome sugere, são construtores de *queries*. Ou seja, a *query* resultante é a mesma do código SQL puro, apenas construída de outra maneira. Tal maneira geralmente

semelhante ao SQL puro, o que implica em um nível de abstração não tão grande sobre o mesmo.

Eles são bibliotecas escritas em linguagens de programação como Python [68] e JavaScript [29], por exemplo. Cada linguagem, portanto, deve ter a sua própria implementação de *Query Builder*. Porém, não é obrigatório que haja apenas uma biblioteca por linguagem. Por conta de geralmente serem bibliotecas de código aberto (*open source*), qualquer pessoa com os devidos conhecimentos pode escrever sua própria biblioteca.

Comumente, a abstração sobre o SQL puro se dá por encadeamento de métodos. Veja como funciona a partir do seguinte código JavaScript escrito utilizando o *Query Builder* Knex.js [31]:

```
knex('users').select('id').where('id', 1).orWhere({votes: 100, country: 'brazil'})
```

O resultado em SQL puro do código acima está representado abaixo:

```
SELECT id FROM users WHERE id = 1 OR (votes = 100 AND country = 'brazil');
```

Percebe-se que realmente o nível de abstração não é tão alto, o que significa que o desenvolvedor não perde tanto controle sobre as operações que estão sendo realizadas no banco de dados. Além disso, como o *Query Builder* compila para SQL puro, a performance é exatamente a mesma.

Algumas vantagens dos *Query Builders* sobre o SQL puro são:

- **extensão de *query*:** a sintaxe de encadeamento de métodos é mais fácil de escrever, estender e reutilizar;
- **injeções SQL:** *Query Builders* têm mecanismos para inserir parâmetros nas *queries* de maneira segura;
- **suporte dos editores de texto:** com a sintaxe de encadeamento de métodos, a *query* deixa de ser interpretada apenas como uma *string* no editor e isso previne erros de digitação, possibilita o realce de sintaxe e o autocompletar (*autocomplete*);
- **suporte a diferentes bancos de dados:** facilita uma possível troca no futuro.

Nessa abordagem, o conhecimento técnico em SQL ainda é necessário e, além disso, existe o conhecimento técnico na biblioteca *Query Builder*. Porém, o desenvolvedor ainda tem flexibilidade: se a funcionalidade desejada não existir no *Query Builder*, ele sempre pode partir para o SQL puro.

2.1.3.3 ORM

Essa é a abordagem com maior nível de abstração de todas. Semelhante aos *Query Builders*, os ORMs também têm grande suporte dos editores de texto. Eles criam um objeto para cada tabela do banco de dados e permitem uma fácil interação entre objetos relacionados, tudo isso com um conhecimento básico (mínimo) de SQL.

A principal vantagem sobre as outras abordagens é que, com ORMs, fazer a gestão de mudanças no banco de dados (como adicionar ou remover novas colunas) é um trabalho relativamente simples e organizado. Existem bibliotecas auxiliares que podem detectar automaticamente alterações nos modelos (tabelas do banco de dados definidas por código) e gerar arquivos de migração (arquivos responsáveis por executar as alterações necessárias no banco de dados).

Porém, tanta abstração traz algumas desvantagens que devem ser levadas em consideração [42]:

- **performance:** ORMs tendem a executar mais *queries* do que o necessário para alcançar o mesmo resultado;
- é tudo "mágica";
- **aprendizado do ORM:** ORMs podem ser complexos de aprender, eles têm muitos recursos e maneiras diferentes de alcançar o mesmo resultado e, geralmente, não é capaz de atender todas as necessidades do desenvolvedor;
- **configuração do ORM:** além de ter que aprender a como realizar *queries* com o ORM, configurar ele da maneira correta pode ser trabalhoso.

2.2 Revisão Bibliográfica

Esta seção tem como objetivo apresentar trabalhos e pesquisas relacionados com o desenvolvimento de aplicações de segurança pessoal e fazer um levantamento das soluções propostas, identificando, principalmente, particularidades e semelhanças entre elas.

2.2.1 Android Personal Safety App

A proposta do trabalho [5] é misturar a segurança pessoal com a segurança geral de uma corporação. Na aplicação definida por ele, existem "administradores" que podem ser corporações de qualquer tipo, como escolas, empresas, delegacias de polícia etc.

As características dessa solução são as que seguem:

- Desenvolvida exclusivamente para plataforma *Android*;
- Conexão com a internet obrigatória para o correto funcionamento;
- Possui botão de pânico;
- *Google Maps* [21] como traçador de rotas entre coordenadas;
- Necessita da inserção de contatos (números de telefone) por parte do usuário;
- Autenticação por meio de *login* e senha;
- Pedido de socorro (pressionamento do botão de pânico) enviado por meio do recurso de notificação do próprio *smartphone*;

- Existência de um *dashboard* para usuários administradores. Em tal *dashboard*, os administradores podem ver o histórico (*log*) de todas as solicitações de ajuda emitidas pelos seus usuários com a possibilidade de aplicação de filtros, como apresentar apenas os pedidos de ajuda ainda não resolvidos, por exemplo;
- O administrador tem a possibilidade de classificar os pedidos de ajuda em resolvidos, não resolvidos ou alarme falso;
- O administrador pode visualizar os dados de seus usuários, inclusive a localização atual dos mesmos;
- Além dos contatos cadastrados pelo usuário, o administrador da organização que o usuário faz parte também é notificado quando ocorre pedidos de socorro;
- Os contatos cadastrados pelo usuário também devem ser usuários do mesmo aplicativo. Ou seja, devem possui-lo instalado em seus *smartphones*;
- Funcionalidade de cancelar um pedido de ajuda, para casos de mal entendido. O cancelamento só é possível se o usuário fizer a confirmação com sua senha de *login*;
- O administrador pode emitir um alerta para todos os seus usuários simultaneamente;
- Um usuário pode ver o histórico de solicitações de ajuda dos seus contatos cadastrados e as emissões de alerta realizadas pelo seu administrador;
- Separa as situações de emergência em categorias (roubo, acidente e perigo);
- O usuário é capaz de ver a própria localização no mapa e, além disso, compartilhar essa informação com outros aplicativos de celular, como *WhatsApp* [71] ou *Instagram* [25].

2.2.2 Womens Safety Mobile App

O trabalho [11] possui as seguintes características:

- Desenvolvida exclusivamente para plataforma *Android*;
- Conexão com a internet obrigatória para o correto funcionamento;
- Possui botão de pânico;
- *Google Maps* [21] como traçador de rotas entre coordenadas;
- Necessita da inserção de contatos (números de telefone) por parte do usuário;
- Autenticação por meio de *login* e senha;
- Pedido de socorro enviado por meio de SMS, em que apenas um dos contatos cadastrados pode ser escolhido pelo usuário para receber uma ligação (e não somente o SMS);

- Como alternativa ao pressionamento do botão de pânico, o usuário pode chacoalhar o celular para cima e para baixo três vezes quando deseja solicitar socorro;
- Quando o socorro é solicitado pelo usuário, o celular começa a vibrar e emitir um barulho estrondoso;
- Possui dois aplicativos separados: um para a vítima e outro para o "guardião";
- Quando em situação de perigo iminente, o usuário pode ativar uma funcionalidade que pede a leitura de sua digital de um em um minuto. Caso seja ultrapassado o período de um minuto sem leitura digital realizada, o aplicativo supõe que o usuário está em perigo e toma as medidas necessárias. Isso contorna o caso em que o usuário não consegue solicitar socorro manualmente (apertando o botão de pânico ou chacoalhando o celular).

2.2.3 Life360

A solução chamada Life360 [23] está ativa no mercado e está disponível tanto para *iOS* quanto para *Android*. Ela se autodenomina "o serviço de segurança familiar líder mundial". Possui funcionalidades tanto para a segurança de localização quanto para a segurança de condução, embora nem todas sejam gratuitas [45].

As principais funcionalidades relacionadas a segurança de localização são as seguintes [17]:

- **Círculo privado:** grupo apenas para convidados em que é possível ver a localização em tempo real um do outro;
- **Histórico de localização:** ver trajetos anteriores de membros do círculo;
- **Alertas de lugares:** notificações ao sair ou chegar em lugares especificados pelos membros do círculo;
- **Zonas personalizadas:** definir zonas em que a localização é mensurada apenas de maneira aproximada;
- **Obter direções com um toque:** sem a necessidade de um endereço, traça a rota até a localização atual do membro do círculo;
- **Alerta SOS:** envia um alerta com a localização atual para os membros do círculo e contatos de emergência;
- **Check-ins:** sinaliza aos membros do círculo que chegou em determinado local.

Já as principais funcionalidades relacionadas a segurança de condução são as seguintes [16]:

- **Resumo de condução em família:** resumo semanal de dados de trânsito do círculo;
- **Relatórios individuais de motorista:** dados de trânsito específicos por usuário;

- **Detalhes de condução:** verifica informações específicas sobre determinada rota, quilômetros registrados, velocidade máxima e quaisquer outros eventos de direção;
- **Monitoramento de velocidade em tempo real:** verificar a velocidade máxima dos membros do círculo enquanto estão em movimento.

2.2.4 SaveMe: a crime deterrent personal safety android app with a bluetooth connected hardware switch

O maior diferencial do trabalho [61] comparado aos demais revisados é a alternativa de ter um botão físico (*hardware*) para solicitar socorro mesmo quando não está com o *smartphone* em mãos. Porém, a conexão com o tal *hardware* pode contribuir para o consumo exagerado de bateria do *smartphone*.

As principais características dessa solução são as seguintes:

- Desenvolvida exclusivamente para plataforma *Android*;
- Conexão com a internet obrigatória para o correto funcionamento;
- Possui botão de pânico;
- *Google Maps* [21] como traçador de rotas entre coordenadas;
- Autenticação por meio de *login* e senha;
- Pedido de socorro enviado por meio de SMS;
- Botão físico (*hardware*) conectado ao *smartphone* por *bluetooth* como maneira alternativa de solicitar socorro;
- A notificação de pedido de socorro é enviada não apenas para os contatos cadastrados pelo usuário mas também para os usuários geograficamente próximos à vítima;
- O solicitante do socorro recebe um *feedback* após algum outro usuário sinalizar que visualizou seu pedido e está a caminho da ajuda;
- Funcionalidade de cancelar um pedido de ajuda, para casos de mal entendido. O usuário define manualmente um número inteiro (na tela de configurações do aplicativo) que ele deve pressionar o botão de pânico para que a solicitação seja cancelada.;
- Os dados de latitude e longitude onde os pedidos de socorro foram realizados são utilizados para análise estatística e definição de áreas geográficas potencialmente perigosas.

Algo que também vale a pena mencionar a respeito desta solução é o fato de que pouco adianta ter acesso à localização das vítimas mas não ter acesso a características como altura, peso, cor dos olhos, cor da pele, cor do cabelo etc. Dessa forma, as únicas pessoas que realmente conseguiriam ajudar seriam os contatos cadastrados (seguindo o pressuposto de que todos os contatos cadastrados sejam pessoas conhecidas do usuário) e não as pessoas geograficamente próximas.

2.2.5 Abhaya: An Android App for the safety of women

A aplicação proposta em [75] é relativamente parecida com a proposta apresentada em [11]. Em ambas há o envio de SMS como forma de alerta e também a possibilidade de ligação para um dos contatos cadastrados. Porém, em [75], não existe nenhuma autenticação de usuário. A particularidade de [75] foi o envio contínuo da localização da vítima em casos de perigo. Os SMSs são enviados de cinco em cinco minutos, atualizando assim a localização atual da vítima, até que ela aperte o botão de parar.

2.2.6 Quadro comparativo

Cada trabalho apresentado contribuiu para a concepção final do aplicativo produzido nesse trabalho. Principalmente o Life360, pois era o mais completo e podia ser facilmente instalado a partir da loja de aplicativos do *smartphone* (os demais não puderam ser instalados ou testados). A tabela 2.1 mostra um comparativo entre as aplicações revisadas e a aplicação proposta, no que tange às funcionalidades.

Tabela 2.1 – Comparação entre as aplicações revisadas e a aplicação proposta

	Aplicação proposta pelo autor	Android Personal Safety App	Womens Safety Mobile App	Life360	SaveMe	Abhaya
Botão de pânico	Sim	Sim	Sim	Sim	Sim	Sim
Botão de pânico físico	Não	Não	Não	Não	Sim	Não
Envio de SMS	Não	Não	Sim	Sim	Sim	Sim
Envio de notificações	Sim	Sim	Não	Sim	Não	Não
Monitoramento de localização em tempo real	Sim	Não	Não	Sim	Não	Não
Lista de contatos	Não	Sim	Sim	Não	Sim	Sim
Diferença de nível entre usuários	Não	Sim	Sim	Não	Não	Não
Grupos independentes entre si	Sim	Sim	Não	Sim	Não	Não
Chat	Sim	Não	Não	Sim	Não	Não
Armazenamento de coordenadas potencialmente perigosas	Sim	Não	Não	Não	Sim	Não
Traçamento de rotas	Não	Sim	Sim	Sim	Sim	Sim
Funcionalidades pagas	Não	Não	Não	Sim	Não	Não

3 Descrição Funcional da Aplicação

Esta seção descreve o modelo funcional da aplicação proposta para estudo.

Seja evitando lugares potencialmente perigosos, rastreando a localização em tempo real de seu ente querido, solicitando ajuda em casos de emergência ou utilizando o *chat* em tempo real para compartilhar informações na forma de texto ou imagem, o importante é que todos os usuários do aplicativo colaborem entre si em prol da segurança.

Os primeiros passos para usar a aplicação são bastante simples e rápidos. Sendo assim, a distância entre o primeiro contato do usuário com o aplicativo até a utilização das funcionalidades existentes não é tão longa. É necessário apenas realizar o cadastro, se autenticar, criar um grupo e convidar pessoas importantes para serem membros.

Dito isso, nas seções subsequentes, serão detalhados a série de funcionalidades e recursos que o aplicativo contém.

3.1 Cadastro de usuário

O usuário fornece alguns dados acerca dele, uma foto de perfil e um nome de usuário (que servirão para outros usuários o identificarem), um email e uma senha (que serão utilizados para autenticação). As capturas de tela referentes a essa funcionalidade são mostradas nas figuras 3.1 e 3.2.

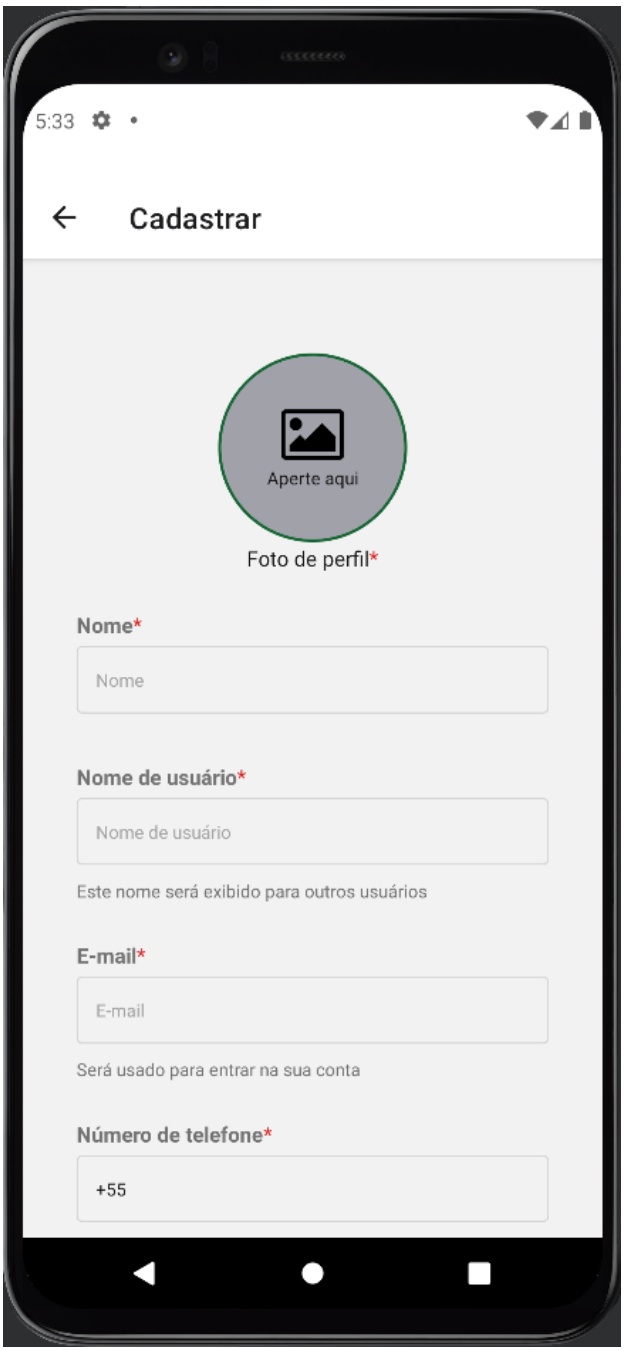

Figura 3.1 – Tela de cadastro.

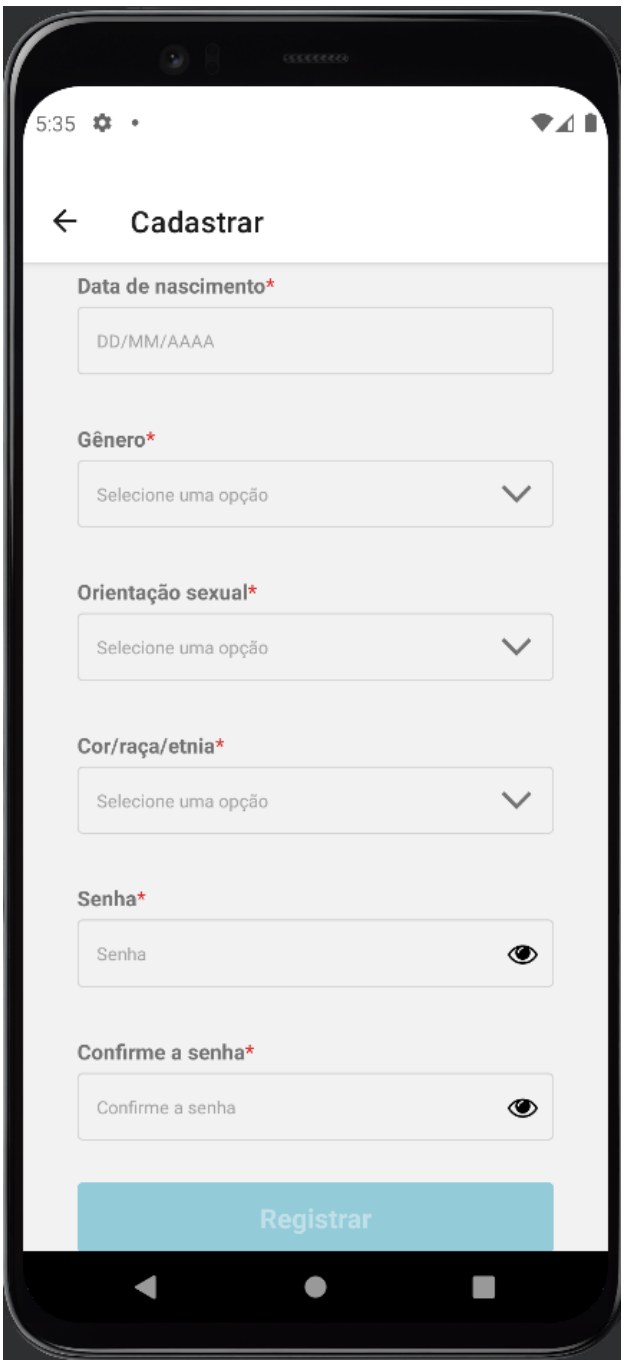

Figura 3.2 – Continuação da tela de cadastro.

3.2 Autenticação

A autenticação do usuário é feita através de email e senha. Se as credenciais estiverem corretas, o usuário tem acesso a todas as funcionalidades e telas do app mobile e também a todas as rotas (antes protegidas) da API. Para proteger as rotas, foi utilizado middlewares do Express e JWT [63].

Para não precisar que o usuário faça o login sempre que abrir o aplicativo, o JWT está sendo salvo no *local storage* do dispositivo e só é removido de lá em caso de logout. Sendo assim, quando

o usuário fecha o aplicativo e abre novamente, ele checa se existe um JWT válido no *local storage*. Em caso positivo, o login é feito automaticamente e em caso negativo, as credenciais são solicitadas novamente.

A tela de login tem links que redirecionam para as telas de *reset* de senha e cadastro. A captura de tela referente a funcionalidade de login é a 3.3.

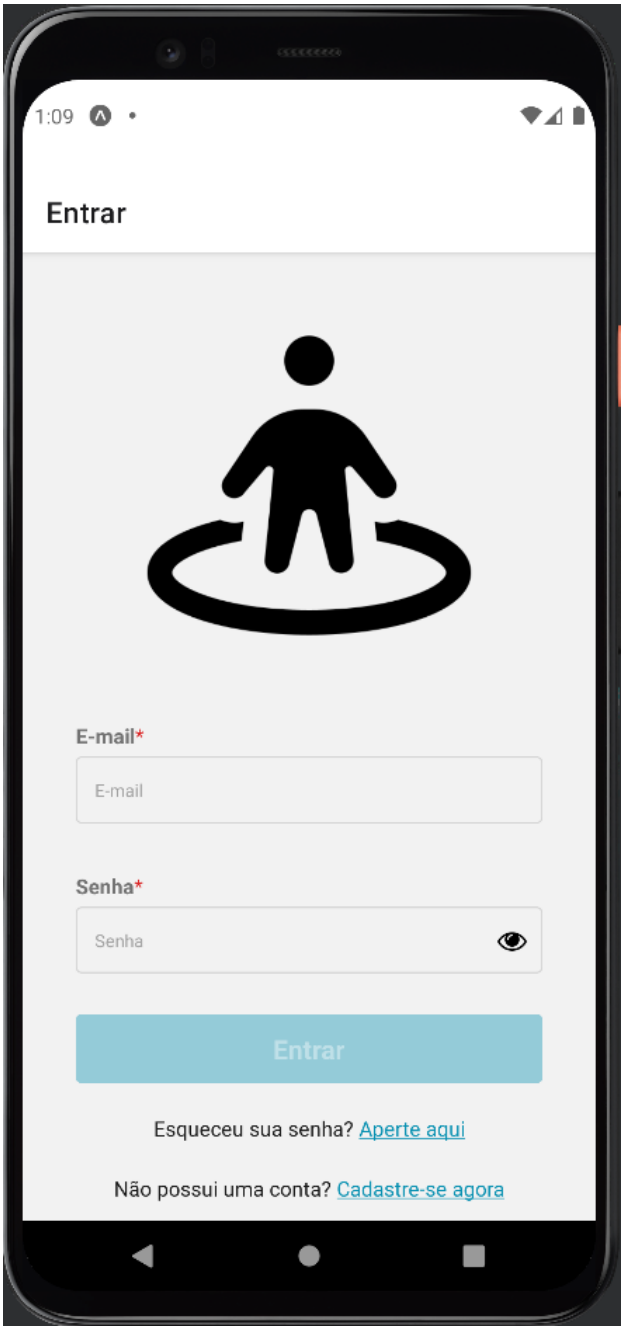

Figura 3.3 – Tela de login.

3.3 Esqueci minha senha

Caso o usuário esqueça sua senha mas lembre de seu endereço de email, é possível gerar uma OTP (*One-Time Password*) para realizar o login, conforme mostra a figura 3.4

Figura 3.4 – Tela de *reset* de senha (criação da OTP).

3.4 Menu inferior

Todas as telas da área logada do usuário possuem o menu inferior mostrado na figura 3.5.

Figura 3.5 – Menu inferior.

Da esquerda para a direita:

- o primeiro item redireciona o usuário para a tela de mapa;
- o segundo item redireciona o usuário para a tela de emergência;
- o terceiro item redireciona o usuário para a tela de notificações;

3.5 Menu superior

Assim como têm um menu inferior, todas as telas da área logada também têm um menu superior. O menu superior tem um item a esquerda, um item central e um item a direita. O item a esquerda redireciona o usuário para a tela de configurações, já o item a direita redireciona o usuário para a tela de *chat*. O item central pode variar de acordo com a tela atual que o usuário se encontra. Quando ele está na tela de *chat* ou na tela de mapa, o item central corresponde a opção de "seleção de grupo" para que o usuário possa alternar entre os grupos que faz parte, conforme mostra a figura 3.6. Isso possibilita uma tela de mapa e uma tela de chat específica para cada grupo. Já nos outros casos, o item central do menu é simplesmente um título que resume a tela atual, conforme mostra a figura 3.7.

Figura 3.6 – Menu superior com *select* de grupos.

Figura 3.7 – Menu superior com título da tela.

Ao abrir a "seleção de grupos", a última opção será um botão que redireciona para a tela de criação de um novo grupo, conforme retrata a figura 3.8.

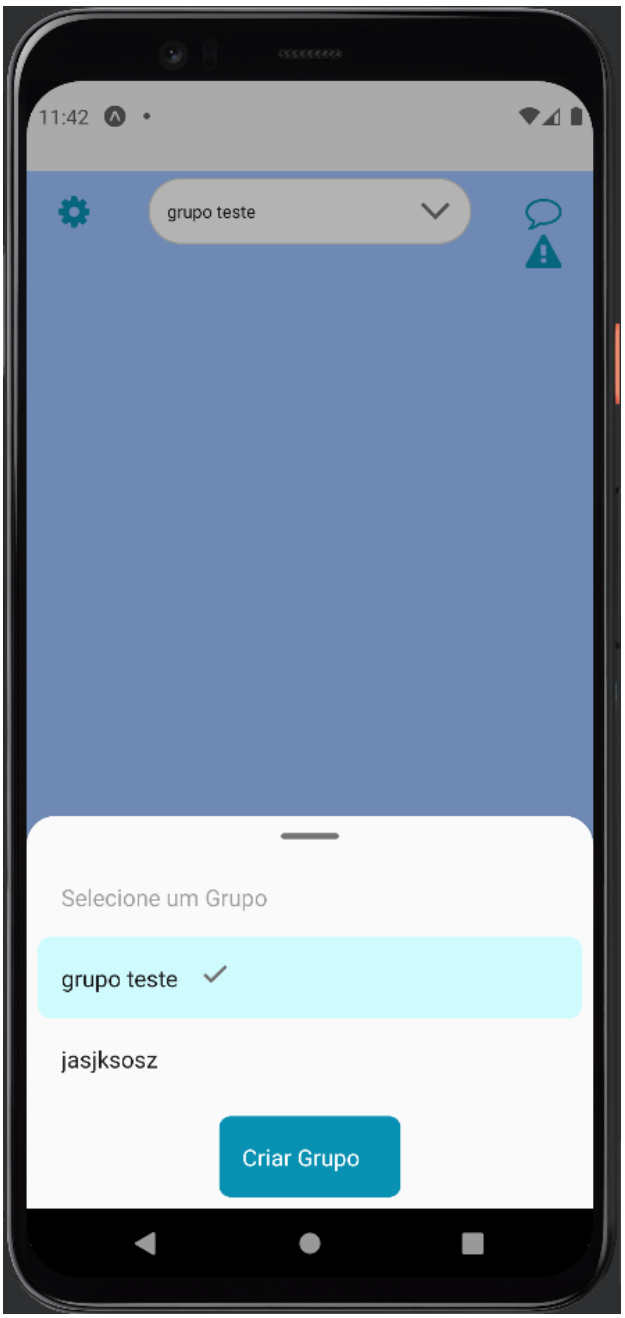

Figura 3.8 – Opções do *select* de grupo e botão para criação de novo.

Ao pressionar o botão, a tela para qual o usuário será redirecionado é a da figura 3.9. Um grupo é composto por seus membros e por seu nome de identificação. Portanto, na tela de criação de grupo, é possível convidar membros e definir um nome para o mesmo. O convite é feito através dos nomes de usuário das pessoas a serem convidadas e o nome do grupo tem um limite de dezesseis caracteres pois a ideia é que seja um nome curto e fácil de lembrar.

Figura 3.9 – Tela de criação de novo grupo.

3.6 Mapa

O mapa tem dois modos, que podem ser alternados entre si ao pressionar o ícone embaixo do ícone de *chat*. O modo padrão é o modo "grupo" (figura 3.10), em que o usuário pode acompanhar a localização em tempo real (ou a última localização conhecida) de outros usuários membros do mesmo grupo. Para alternar entre grupos, basta abrir as opções do *select* que se encontra no centro superior da tela. O outro modo do mapa é o modo "áreas de risco" (figuras 3.11 e 3.12). Após a realização do login, a tela de mapa no modo "grupo" é a primeira tela que o usuário tem contato.

Quanto menor for o *zoom* do mapa, mais as áreas de risco vão se agrupando até que se tornem

somente um ponto mapa e quanto maior for o *zoom*, o contrário acontece: as áreas de risco vão se desagrupando e se tornando mais específicas. A figura 3.11 mostra um *zoom* considerável que abrange toda a região do Distrito Federal enquanto que a figura 3.12 mostra apenas uma parte da região administrativa de Ceilândia.

Aqui, áreas de risco são todas as coordenadas em que algum usuário do aplicativo solicitou ajuda. Solicitar ajuda corresponde ao usuário pressionar o botão presente na tela de emergência, que será especificada posteriormente.

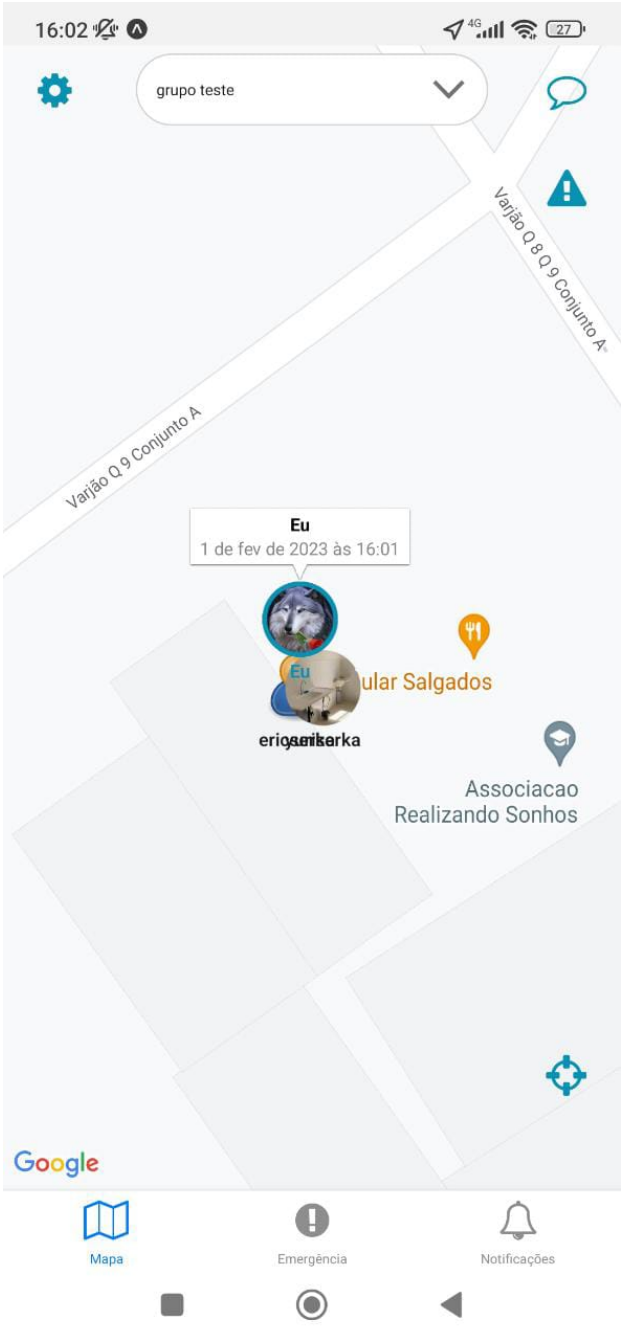

Figura 3.10 – Mapa no modo “grupo”.

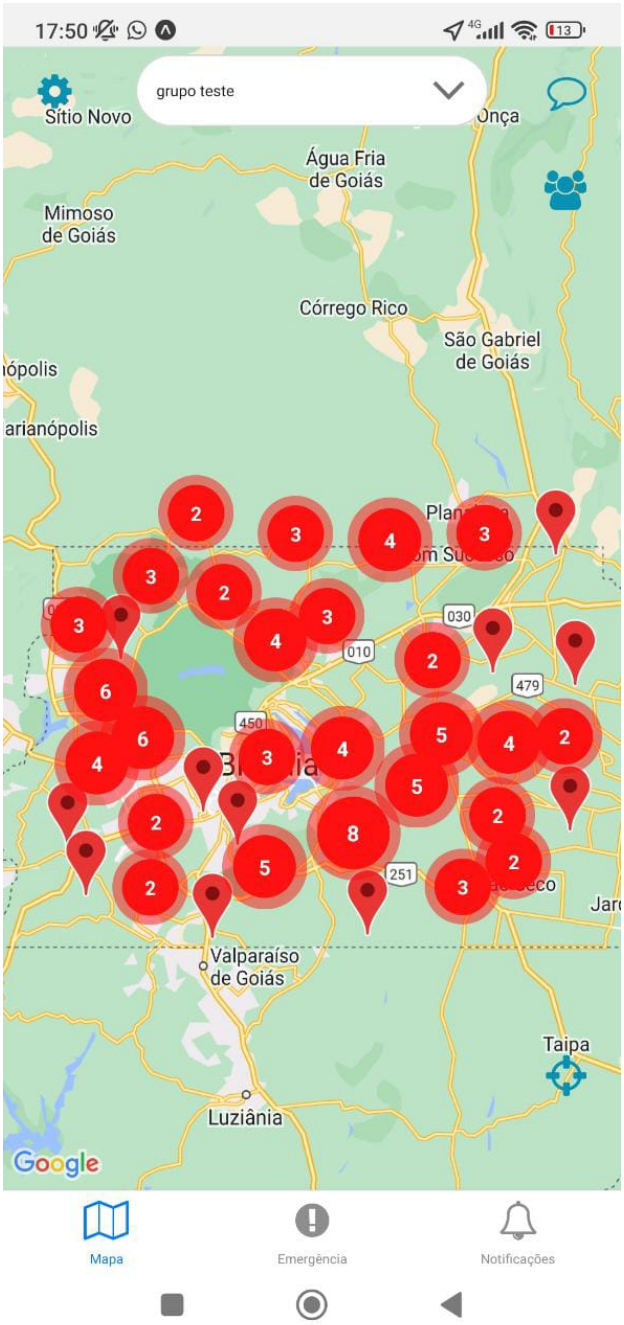

Figura 3.11 – Mapa no modo “áreas de risco” com um *zoom* menor.

Figura 3.12 – Mapa no modo “áreas de risco” com um *zoom* maior.

3.7 Emergência

Tela com apenas um botão que envia uma notificação de solicitação de ajuda para todos os membros de todos os grupos que o usuário faz parte. Captura de tela: figura 3.13.

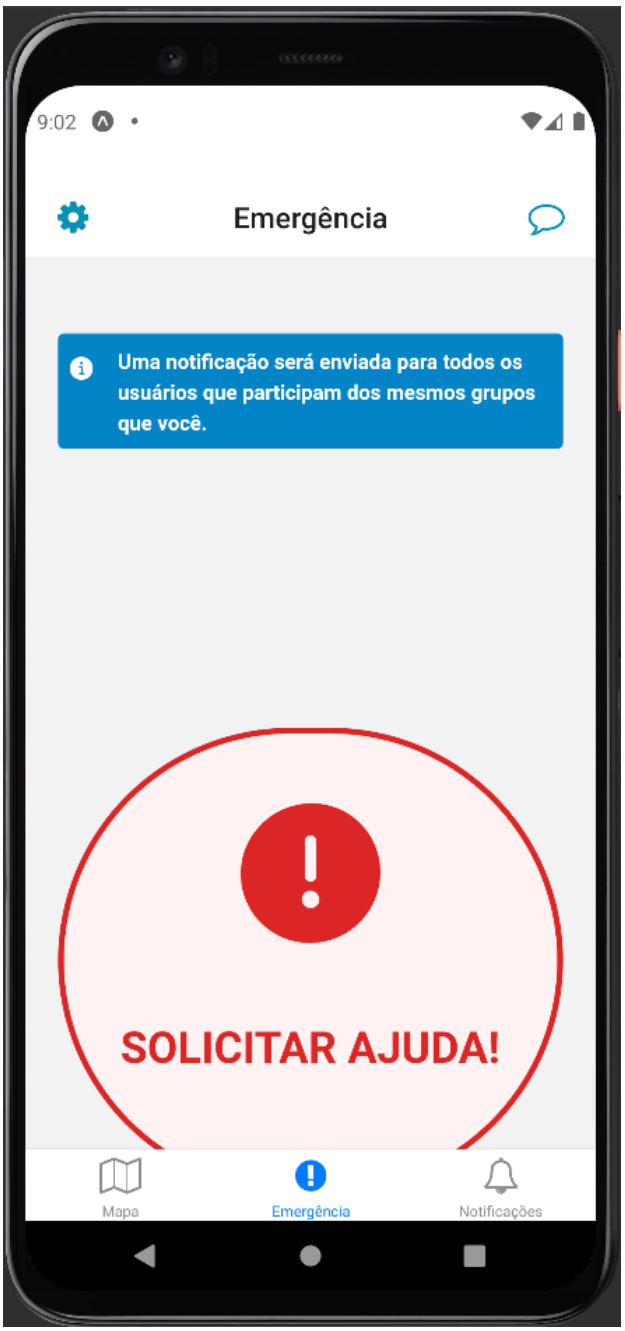

Figura 3.13 – Tela de emergência.

3.8 Notificações

Tela com histórico de todas as notificações que o usuário recebeu, sendo as com opacidade menor (um pouco transparentes) as que já foram visualizadas. Existem quatro tipos de notificações lidadas pela aplicação:

1. Convites para ingresso em grupos;
2. Solicitações de ajuda;

3. Novas mensagens nos chats de grupo;

4. Avisos de remoção do grupo (membro removido ou grupo excluído).

Além disso, para cada notificação salva no banco de dados da aplicação, uma *notificação push* [70] é emitida para os dispositivos correspondentes. Um exemplo de *notificação push* pode ser visualizado na figura 3.14. As *notificações push* são emitidas pelo servidor graças a biblioteca *expo-server-sdk* [14] do Node.js.

A captura de tela da tela de notificações pode ser visualizada na figura 3.15.

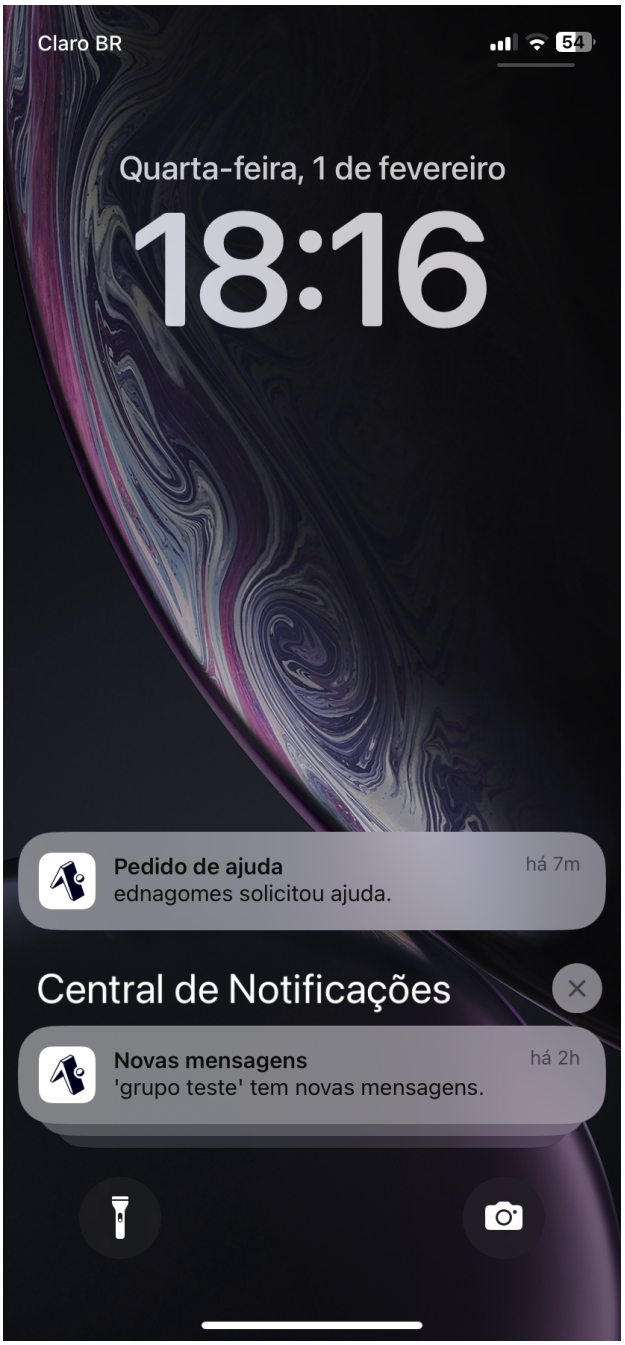

Figura 3.14 – Exemplo de *notificações push*.

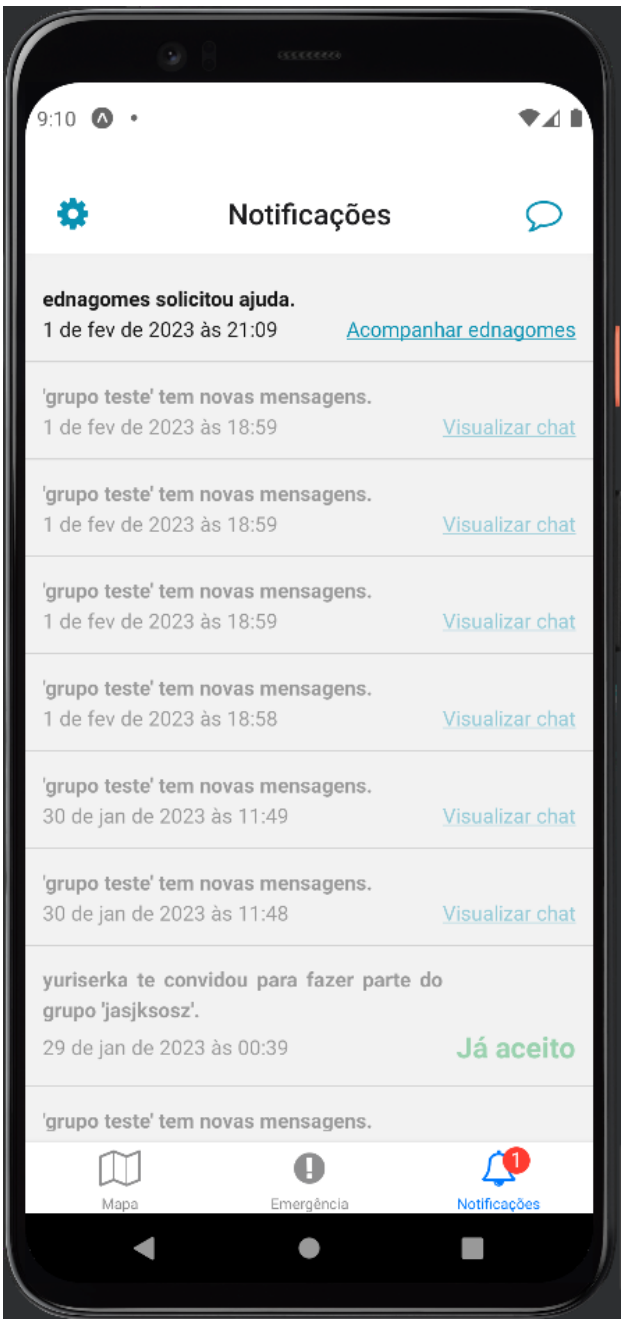

Figura 3.15 – Tela de notificações.

3.9 Chat

Existe um chat para cada grupo que o usuário participa, sendo a seleção do grupo feita no *select* no centro superior da tela. No *chat*, é possível enviar textos e/ou imagens para multiplos usuários simultaneamente. Captura de tela: figura 3.16

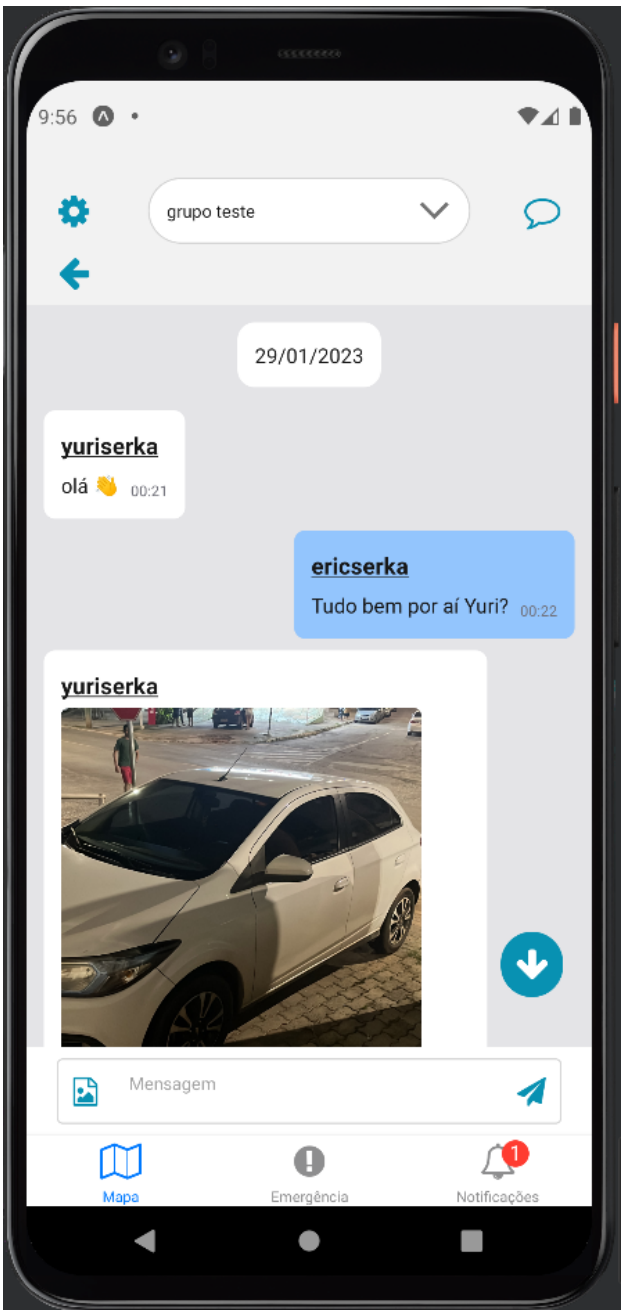

Figura 3.16 – Tela de *chat*.

3.10 Configurações

Seção onde o usuário faz alterações nos dados do seu perfil, escolhe com quais grupos quer compartilhar sua localização e faz gerenciamento dos grupos que participa

3.10.1 Editar dados

Semelhante a tela de cadastro, só que aqui com os dados já previamente preenchidos, é um formulário em que o usuário pode editar os dados de seu perfil/conta. Capturas de tela: figura 3.17, 3.18 e 3.19.

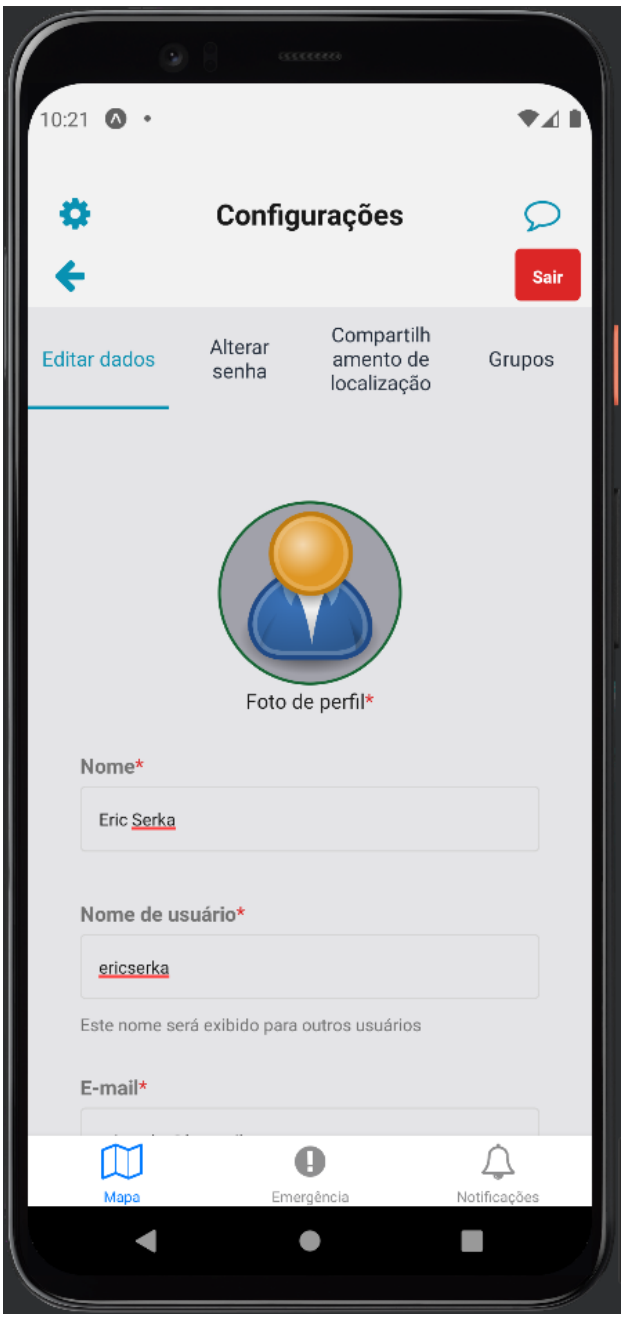

Figura 3.17 – Tela de edição dos dados pessoais - primeira parte.

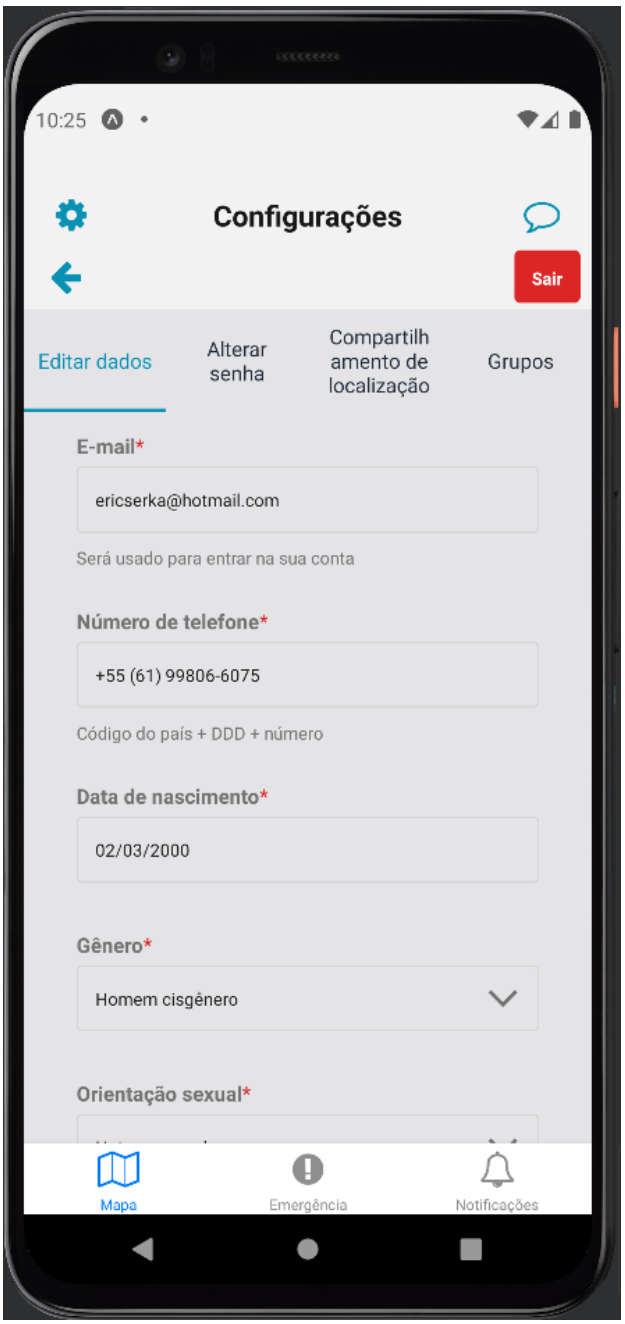

Figura 3.18 – Tela de edição dos dados pessoais - segunda parte.

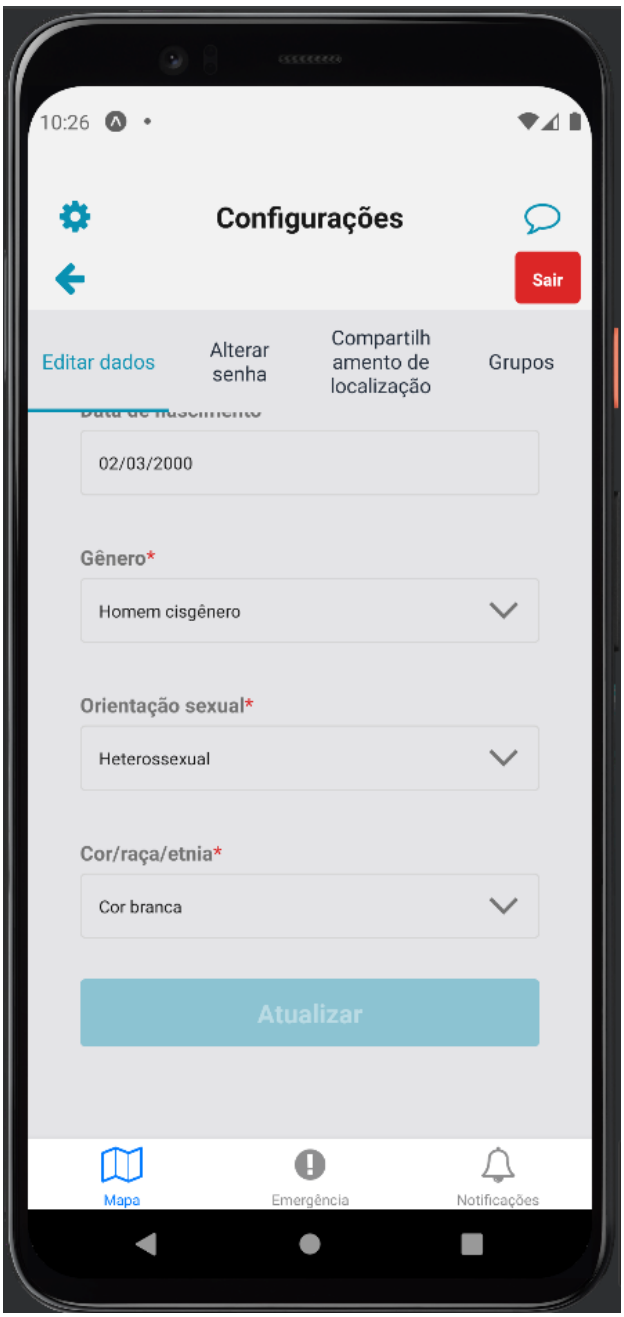

Figura 3.19 – Tela de edição dos dados pessoais - terceira parte.

3.10.2 Alterar senha

Aqui é possível alterar a senha antiga para uma nova. Captura de tela: 3.20

Figura 3.20 – Tela de alteração de senha.

3.10.3 Compartilhamento de localização

Tela que possibilita ao usuário selecionar quais grupos que ele pertence devem receber atualizações de sua localização. Se o *switch* estiver ligado, quer dizer que aquele grupo vai receber atualizações caso sua localização mude e, caso contrário, o grupo não receberá atualizações. Captura de tela: figura 3.21.

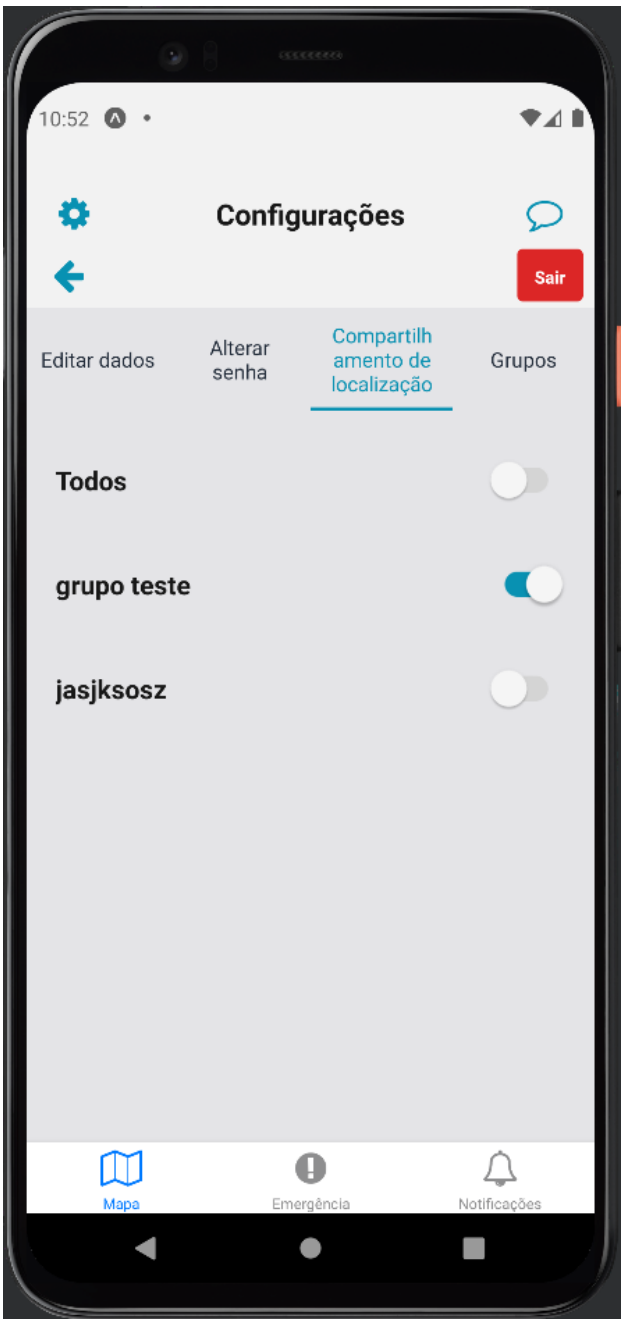

Figura 3.21 – Tela de compartilhamento de localização.

3.10.4 Grupos

Tela que lista os grupos que o usuário faz parte (figura 3.22). Cada grupo possui uma tela de configurações. Configurações essas que variam caso o usuário seja o dono do grupo em questão ou se ele é apenas um membro. Porém, percebe-se que uma parte comum que independe do nível do usuário em relação ao grupo é a definição de tal grupo como o padrão ou não, como pode ser visto nas figuras 3.23 e 3.25. Grupo padrão é aquele que é carregado por padrão no mapa em modo "grupo" quando o aplicativo é inicializado. Sendo assim, não é possível ter dois ou mais grupos definidos como padrão, apenas um.

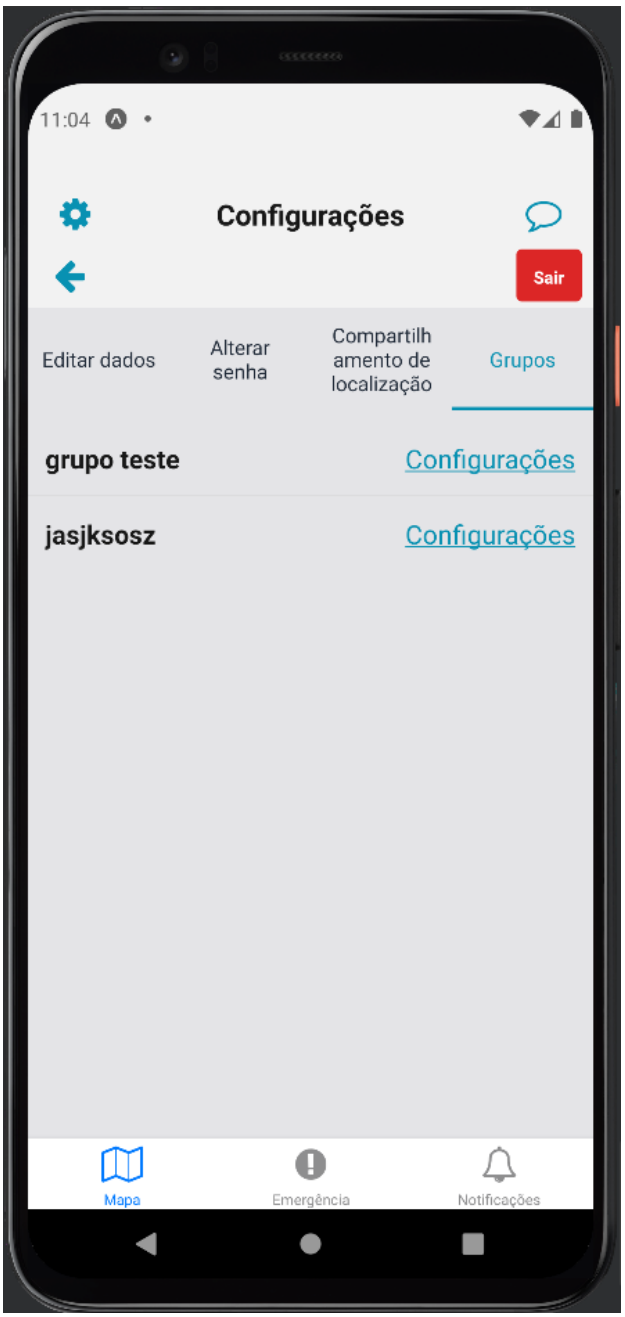

Figura 3.22 – Tela de listagem de grupos.

3.10.4.1 Configurações de grupo criado pelo usuário

Nos casos em que o usuário é dono do grupo, ele tem as opções de (conforme reflete a figura 3.23):

1. convidar mais pessoas para serem membros;
2. remover usuários específicos;
3. excluir o grupo por completo e, consequentemente, remover todos os membros do mesmo.

A tela da figura 3.24 possibilita ao usuário convidar novos membros ao seu grupo, sendo a única coisa necessária o nome de usuário da pessoa a ser convidada.

Com o convite enviado, cabe ao usuário convidado decidir se aceita entrar no grupo ou não (funcionalidade esta presente na tela de notificações).

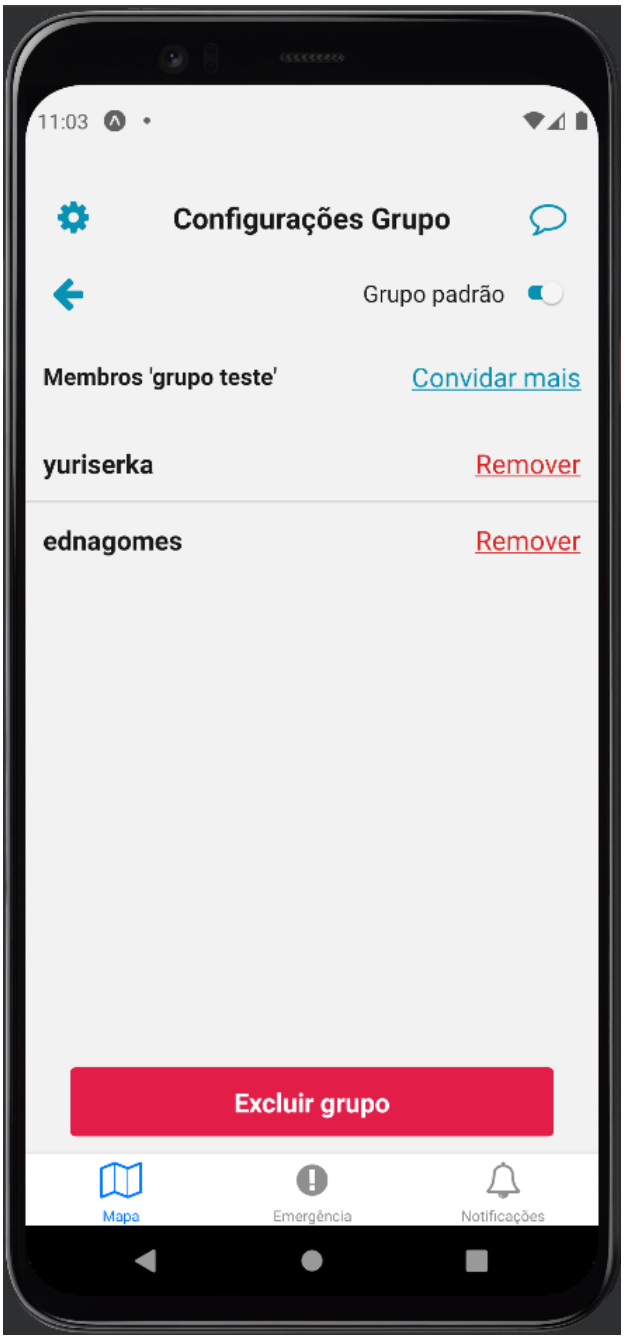

Figura 3.23 – Tela de configuração de grupo criado pelo usuário.

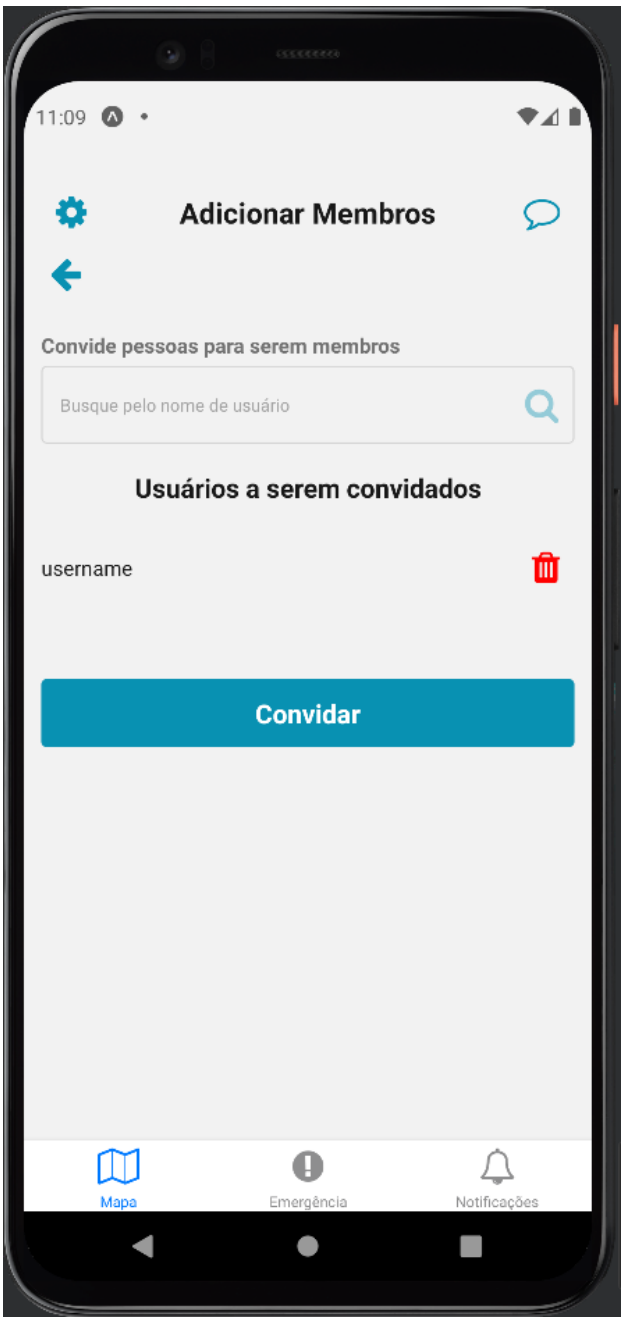

Figura 3.24 – Tela para convidar novos membros para o grupo.

3.10.4.2 Configurações de grupo que usuário é apenas membro

Nos casos em que o usuário é apenas membro do grupo, a única opção que ele tem é sair do mesmo, conforme mostra a figura 3.25.

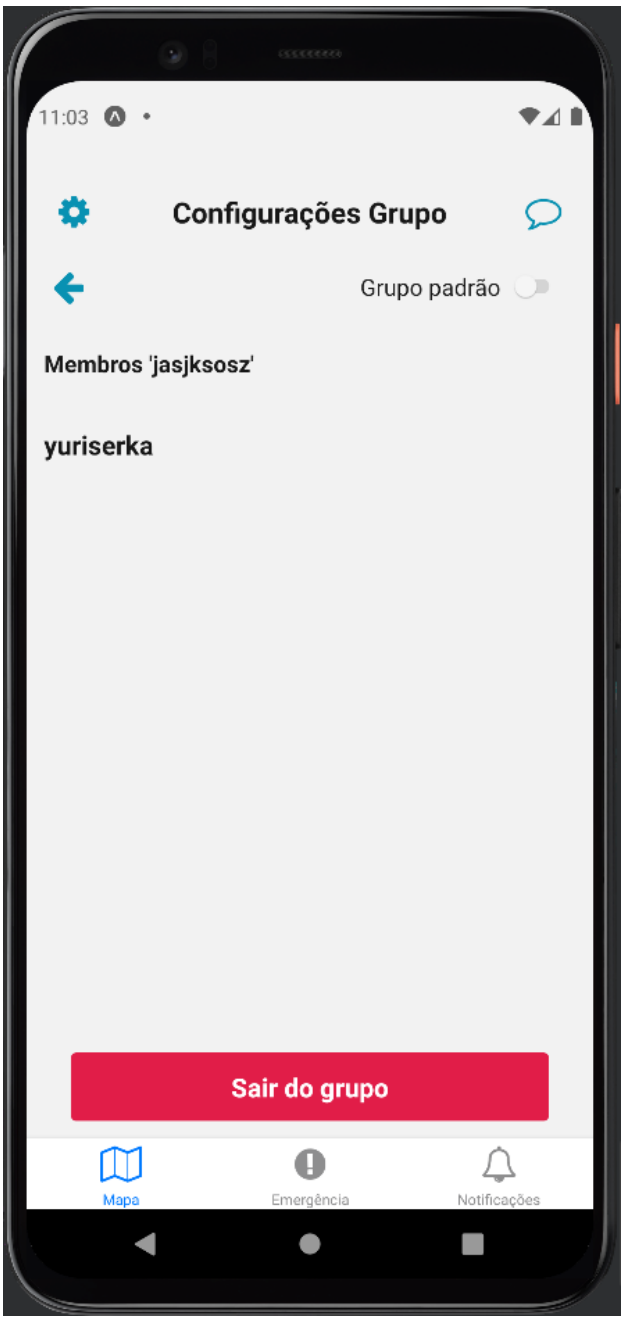

Figura 3.25 – Tela de configuração de grupo em que usuário é apenas membro.

3.10.5 Botão sair

É o botão vermelho presente em todas as abas da seção de configurações. Ao pressioná-lo, a ação de *logout* é executada. Ação esta que possui os seguintes passos:

1. remover o JWT do *local storage* (lado do cliente).
2. remover o *push token* do atual usuário do banco de dados (lado do servidor);

O primeiro passo previne um futuro *login* automático inapropriado já que o *local storage* estar vazio significa que o usuário terá de se autenticar novamente. Já o segundo passo previne que

o usuário receba *notificações push*, já que ter um *push token* válido é pré-requisito para receber *notificações push*. O *push token* é um token anônimo que representa um dispositivo, é ele quem diz para o Expo qual dispositivo deve receber a notificação. Atualizando o valor desse *token* para nulo no logout impossibilita, assim, o recebimento de futuras *notificações push*.

4 Implementação e Resultados

As soluções da aplicação foram desenvolvidas com o máximo de simplicidade mas de maneira contínua. A todo momento testes, revisões e refatorações aconteceram, o que acarretou em um dinamismo ativo ao projeto.

As próximas seções fornecem mais detalhes técnicos acerca de cada componente que compõe a aplicação final, além dos resultados obtidos.

4.1 Tecnologias e ferramentas

A linguagem de programação escolhida para o desenvolvimento dos componentes foi a *JavaScript* [29] pelos seguintes motivos:

- familiaridade do autor com a linguagem;
- vasta comunidade ativa;
- vasta quantidade de bibliotecas de código aberto disponíveis;
- aplicativo móvel e servidor escritos na mesma linguagem.

A principal ferramenta utilizada, comum ao *frontend* (aplicativo móvel) e ao *backend* (servidor) (por conta de ambos projetos terem sido escritos em *JavaScript*), foi o *Yarn* [74]. Esta ferramenta é um gerenciador de pacotes e bibliotecas para projetos do ecossistema *JavaScript* alternativo ao padrão, que é o NPM (*Node Package Manager*) [39]. É esta ferramenta que possibilita a instalação, remoção e atualização de dependências no projeto. Dependências estas que serão mencionadas a seguir.

Por se tratarem de coisas fundamentalmente diferentes, as dependências do aplicativo móvel não se assemelham às dependências do servidor. Por esse motivo, serão abordadas em seções distintas.

Porém, uma dependência comum para o lado do cliente e o lado do servidor é o *Socket.IO*, uma biblioteca para comunicação bidirecional, de baixa latência e baseada em eventos que foi utilizada na aplicação para atualizações em tempo real de localização, notificações, etc. *Socket.IO* foi construído sobre o protocolo *WebSocket* mas com garantias adicionais, como *fallback* para HTTP *long-polling* e reconexão automática [54]. Ou seja, provê maneiras de ter conexão persistente com o servidor.

4.1.1 Frontend - Aplicativo móvel

Como principal tecnologia para o desenvolvimento da componente do aplicativo móvel, foi utilizada a biblioteca *React Native* [49]. Escrita em *JavaScript* e renderizada com código nativo,

com o React Native, é possível desenvolver aplicativos móveis para *Android* e *IOS* utilizando apenas um código fonte.

Outra tecnologia fundamental foi o *Expo* [12]. Com ela, foi possível testar o aplicativo enquanto ele ainda estava sendo desenvolvido graças ao aplicativo *Expo Go* [13], que está disponível tanto na *App Store* do *iOS*, quanto na *Google Play* do *Android*. Graças ao *Expo*, os *softwares Xcode* [73] e *Android Studio* [4] não foram necessários, softwares de desenvolvimento *iOS* e *Android*, respectivamente.

Expo é um conjunto de ferramentas construídas sobre o *React Native*. No contexto de desenvolvimento do presente aplicativo, ele permitiu:

- gerenciar as permissões de localização, notificação e acesso a galeria;
- customizar a barra de status;
- criar uma tela de inicialização, que permanece visível enquanto o aplicativo está sendo carregado;
- o envio e recebimento de *notificações push*.

Para a comunicação com o servidor, foram utilizadas as bibliotecas *Axios* [6], um cliente HTTP baseado em promessa que usa *XMLHttpRequests*, para consumo da API e *socket.io-client* [56] para emissão e recebimento de eventos no *socket* no lado do cliente.

4.1.2 Backend

O servidor foi implementado utilizando como tecnologia o *Express*, um *framework* web minimalista e rápido para o Node.js, um ambiente *JavaScript* que permite desenvolvedores escreverem programas no lado do servidor fora de um navegador [15].

Com o *Express*, foi possível implementar as rotas (*endpoints*) da API a ser consumida pelo aplicativo móvel. Algumas dessas rotas, as que só deveriam ser consumidas no contexto de usuário logado e autenticado, foram protegidas graças a combinação de *middlewares* do *Express* e JWT. O fluxo de uma requisição protegida pode ser visulizada na figura 4.1.

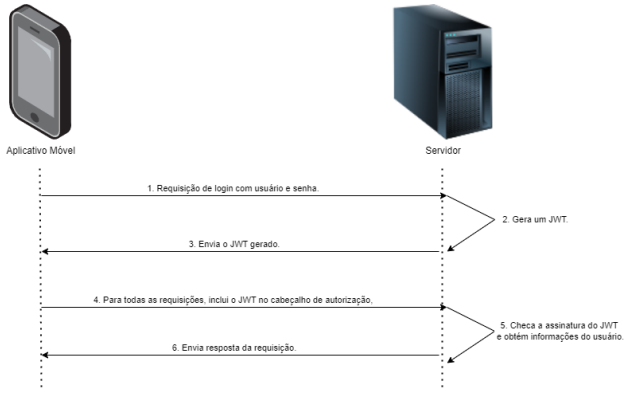

Figura 4.1 – Autenticação JWT - Fluxo. Extraído e adaptado de [30]

Importante salientar que o servidor só gera o JWT nos casos em que usuário e senha forem fornecidos corretamente. Caso contrário, o aplicativo móvel recebe como resposta uma mensagem do tipo "Não autorizado".

Tanto para a criação e quanto para a verificação da assinatura de um JWT, o servidor utiliza de um segredo, definido como uma variável de ambiente de tipo texto.

Como camada de dados, foi utilizado o ORM *Prisma* [48]. Ele foi utilizado como ferramenta para realizar todas as consultas com o banco de dados, assim como modelar o mesmo. Além disso, ele foi utilizado para controlar e versionar a evolução do banco de dados, salvando o histórico de mudanças e possibilitando o retorno para uma determinada versão (ou determinado estado). Conceito esse denominado como *Database Migrations* ou *Schema Migrations*. Cada "migration" contém todas as mudanças realizadas no banco de dados de uma versão para outra. Portanto, contém códigos SQL como aqueles que criam tabela, adicionam ou removem coluna, alteram o nome de uma tabela ou coluna, etc.

Por fim, para que o aplicativo móvel se conecte ao *socket.io* [55] a partir do *socket.io-client*, o servidor tem que estar disponível em determinada porta. Isso foi possível graças ao módulo "http" do *Node.js*, que consegue criar um servidor com uma aplicação *Express* e *socket.io* simultaneamente.

4.1.3 Banco de dados

Como a integridade das informações e a consistência nas relações entre as entidades participantes da aplicação eram uma prioridade, um banco de dados relacional foi escolhido para compor o projeto. Por conta de familiaridade do autor, foi escolhido o *MySQL* [35], mas existem outras boas opções como o *PostgreSQL* [46] e *MariaDB* [33] ou *SQLServer* [60].

O diagrama entidade relacionamento do banco de dados pode ser visualizado na figura 4.2. Atenção especial para as tabelas *_groupLocationShared* e *_groupMembers* que são as chamadas tabelas de associação. Elas são responsáveis por criar dois relacionamentos de um para muitos (um entre cada uma das duas tabelas associadas) [50]. No caso, as duas tabelas associadas são *Group* e *User*, mas as associações têm significados diferentes: a primeira representa quais grupos os usuários determinam que desejam compartilhar sua localização e a segunda representa a totalidade de membros dos grupos.

Outra tabela importante é a *_prisma_migrations*, que corresponde ao histórico de todas as *migrations* que foram aplicadas no banco de dados até o presente momento, em ordem cronológica pois a ordem é importante.

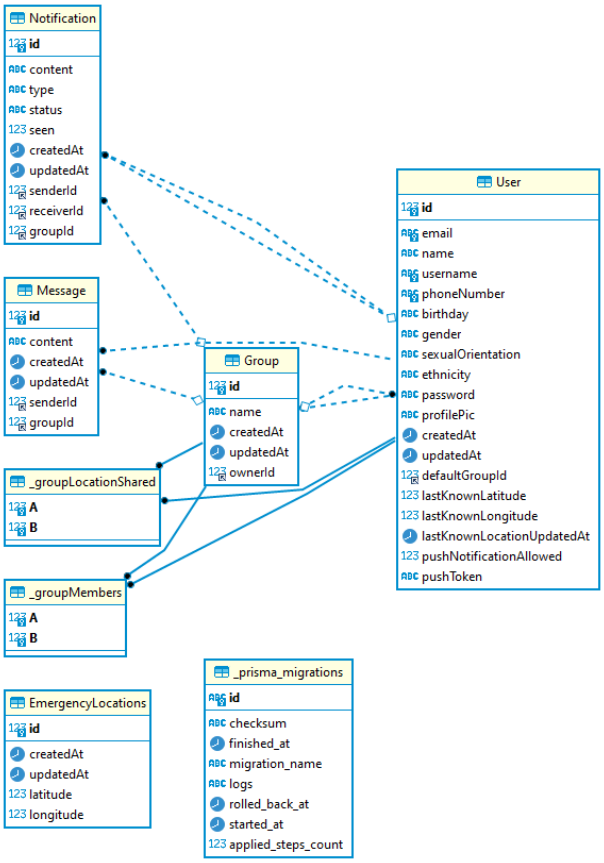

Figura 4.2 – Diagrama entidade relacionamento.

4.2 Arquitetura

A arquitetura da aplicação pode ser visualizada na figura 4.3. Ela é composta basicamente por três componentes: o aplicativo móvel, o servidor e o banco de dados.

Sendo que:

- o aplicativo móvel é composto por:
  - *React Native*
  - *EXPO*
  - *Socket.io*
  - outras bibliotecas de código aberto desenvolvidos pela comunidade *JavaScript*
- e o servidor composto por:
  - *Node.js*
  - *Express*
  - *Socket.io*
  - *Prisma*

– outras bibliotecas de código aberto desenvolvidos pela comunidade *JavaScript*

Como exemplo de bibliotecas de código aberto impactantes no desenvolvimento da aplicação, tem-se o *NativeBase* [37] e o *bcrypt* [7]. A primeira é uma biblioteca de componentes que auxiliou na criação das interfaces de usuário (*User Interface* - UI) do aplicativo móvel e a segunda é uma biblioteca que permitiu realizar a criptografia e decriptografia das senhas dos usuários.

O *NativeBase*, além de fornecer componentes como botões, modais, menus, etc., também permitiu a estilização das UIs do aplicativo. Tudo isso contribuiu para um desenvolvimento mais rápido e produtivo das telas. Já o *bcrypt* possibilitou o não salvamento das verdadeiras senhas dos usuários no banco de dados, o que está de acordo com a LGPD - Lei Geral de Proteção de Dados Pessoais. Invés disso, é o *hash* criptografado da senha que é salvo e, para realizar a autenticação (login), esse *hash* é decriptografado e comparado com o valor fornecido pelo usuário.

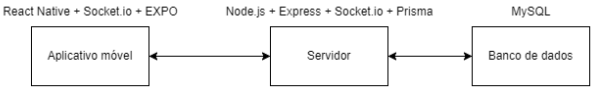

Figura 4.3 – Arquitetura da aplicação.

4.3 Resultados

A arquitetura da aplicação final resultante é bastante simples e intuitiva, como pôde ser visto na figura 4.3. A codificação, tanto do *backend* quanto do *frontend*, foi feita utilizando apenas uma linguagem de programação, a *JavaScript*. Bibliotecas atualizadas, populares, de código aberto e com boa documentação foram utilizadas e, além disso, boas práticas de programação foram adotadas como a escrita de código limpo e legível.

A aplicação foi desenvolvida buscando sempre o melhor tratamento de erro e o melhor controle de estado possível a fim do usuário ter uma boa experiência e não ter inconsistência nos dados visualizados, além de ter sido projetada para ter uma boa disponibilidade.

Apesar da aplicação não ter apresentado gargalos com testes realizados com poucos usuários simultâneos, o cenário muda quando o número de usuários aumenta. Tal impacto na usabilidade da aplicação por conta de lentidão ou afins com certeza seria algo a melhorar na aplicação e totalmente cabível de ser um trabalho futuro.

O código utilizado na criação dos componentes que compõe a aplicação foi versionado por meio do sistema de controle de versão Git e está hospedado na plataforma GitHub. O repositório encontra-se disponível publicamente através da URL <https://github.com/ericserka/pfg2>.

5 Conclusões e Trabalhos Futuros.

Este trabalho teve como objetivo realizar um estudo dos componentes que compõe uma aplicação de segurança colaborativa baseada em geolocalização.

Com o intuito de criar uma base sólida de conhecimento, antes de iniciar o desenvolvimento da aplicação em si, foram pesquisados aspectos teóricos envolvidos e foram revisados trabalhos anteriores com propostas semelhantes. Isso foi fundamental para a compreensão geral dos componentes envolvidos (aplicação móvel, servidor e banco de dados), para a decisão do autor na escolha das tecnologias a serem utilizadas e na análise do resultado final da aplicação.

Na fase de implementação da aplicação, foi realizado um levantamento dos requisitos mínimos necessários para uma aplicação apresentável e viável seguido da organização das tarefas. Após isso, foi iniciada a fase de codificação da aplicação de fato.

Com a análise da aplicação final, que pode ser observada nos capítulos 3 e 4, pode-se afirmar que o objetivo do estudo foi satisfatório quando comparado com os objetivos propostos no capítulo inicial. No decorrer do estudo proposto foi possível concluir como as características de um sistema influenciam na usabilidade do mesmo e como fatores como a escolha das tecnologias ideais podem influenciar nas características finais de um sistema. Foi possível visualizar como as decisões de um projetista são importantes para a garantia de qualidade da aplicação final e como um estudo minucioso deve ser realizado a fim de evitar futuros problemas.

A aplicação final desenvolvida nesse trabalho é um MVP (*Minimum Viable Product* - Produto Viável Mínimo). Portanto, algumas outras funcionalidades poderiam ser implementadas para um melhor cumprimento do objetivo de promover a segurança colaborativa a partir de aplicativos móveis e geolocalização. Como sugestões de trabalhos futuros tem-se:

- Implementação de *Logs* de histórico de coordenadas para visualizar as rotas que os usuários traçaram;
- Implementação de funcionalidade de análise e visualização de dados para analisar o perfil dos usuários que solicitaram emergência e analisar as coordenadas que possam ser consideradas mais perigosas;
- Implementação de um *widget* (atalho) para apertar botão de pânico (solicitação de emergência) mais facilmente;
- Refatoração do que está desenvolvido atualmente, como por exemplo melhorar a componentização das telas do *React Native* e se atentar mais com problemas de performance (principalmente àqueles relacionados à "rerenderização" de elementos);
- Implementação de um controle de nível dos membros do grupo: membros do mesmo grupo com níveis e permissões diferentes podem possibilitar cenários como o de controle parental, por exemplo;

- Implementação de locais personalizados por grupo: possibilidade de cada grupo ter locais favoritos e os membros serem notificados quando certo membro chega ou sai de determinado local;
- Implementação de monitoramento de mudanças de localização mesmo com o aplicativo móvel aberto em segundo plano no *smartphone*.

Bibliografia

[1] *101 Switching Protocols*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status/101> (acesso em 25/10/2022).

[2] *About Objective-C*. URL: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html> (acesso em 30/10/2022).

[3] *Android | A plataforma que redefine o impossível*. URL: <https://www.android.com/intl/pt-BR_br/> (acesso em 26/09/2022).

[4] *Android Studio*. URL: <https://developer.android.com/studio> (acesso em 06/02/2023).

[5] S ARTHI e Mr K Nirmal. "Android Personal Safety App". Em: *Journal homepage: www. ijrpr. com ISSN* 2582 (), p. 7421.

[6] *Axios*. URL: <https://axios-http.com/> (acesso em 06/02/2023).

[7] *bcrypt*. URL: <https://www.npmjs.com/package/bcrypt> (acesso em 08/02/2023).

[8] *Cross Site Request Forgery (CSRF)*. URL: <https://owasp.org/www-community/attacks/csrf> (acesso em 25/09/2022).

[9] *CSS*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/CSS> (acesso em 30/10/2022).

[10] *Dart programming language*. URL: <https://dart.dev/> (acesso em 30/10/2022).

[11] Kanakaveti Narasimha Dheeraj, Goutham RJ et al. "Womens Safety Mobile App". Em: *International Journal on Cybernetics & Informatics (IJCI)* 10.1 (2021), p. 2.

[12] *Expo*. URL: <https://expo.dev/> (acesso em 06/02/2023).

[13] *Expo Go*. URL: <https://expo.dev/client> (acesso em 06/02/2023).

[14] *expo-server-sdk - npm*. URL: <https://www.npmjs.com/package/expo-server-sdk> (acesso em 01/02/2023).

[15] *Express*. URL: <http://expressjs.com/> (acesso em 06/02/2023).

[16] *Features: Driving Safety - Life360*. URL: <https://www.life360.com/intl/intl-features-driving-safety/> (acesso em 30/01/2023).

[17] *Features: Location Safety - Life360*. URL: <https://www.life360.com/intl/intl-features-location-safety/> (acesso em 30/01/2023).

[18] *Flutter - Build apps for any screen*. URL: <https://flutter.dev/> (acesso em 30/10/2022).

[19] *Gerencie os projetos do time em qualquer lugar | Trello*. URL: <https://trello.com/pt-BR> (acesso em 11/02/2023).

[20] *GitHub*. URL: <https://github.com/> (acesso em 11/02/2023).

[21] *Google Maps*. URL: <https://www.google.com.br/maps/> (acesso em 26/09/2022).

[22] *Google Meet*. URL: <https://meet.google.com/> (acesso em 11/02/2023).

[23] *Homepage - Life360*. URL: <https://www.life360.com/intl/> (acesso em 30/01/2023).

[24] *HTML: Linguagem de Marcação de Hipertexto*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/HTML> (acesso em 30/10/2022).

[25] *Instagram*. URL: <https://www.instagram.com/> (acesso em 26/09/2022).

[26] *Instruções de direção, atualizações do tráfego em tempo real condições das estradas - Waze*. URL: <https://www.waze.com/pt-BR/live-map/> (acesso em 11/02/2023).

[27] *iOS 16 - Apple (BR)*. URL: <https://www.apple.com/br/ios/ios-16/> (acesso em 29/10/2022).

[28] *Java | Oracle*. URL: <https://www.java.com/pt-BR/> (acesso em 30/10/2022).

[29] *JavaScript*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript> (acesso em 30/10/2022).

[30] *JWT tokens and security – working principles and use cases*. URL: <https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/> (acesso em 07/02/2023).

[31] *Knex.js*. URL: <https://knexjs.org/> (acesso em 07/11/2022).

[32] *Kotlin Programming Language*. URL: <https://kotlinlang.org/> (acesso em 30/10/2022).

[33] *MariaDB*. URL: <https://mariadb.org/> (acesso em 08/02/2023).

[34] *Modos de transmissão de dados: Simplex, Half-Duplex, Full-Duplex*. URL: <https://www.canalti.com.br/redes-de-computadores/modos-de-transmissao-de-dados-simplex-half-duplex-full-duplex/> (acesso em 23/09/2022).

[35] *MySQL*. URL: <https://www.mysql.com/> (acesso em 08/02/2023).

[36] *Native vs Hybrid: Which Mobile App Platform Should You Choose?* URL: <https://www.codemotion.com/magazine/frontend/mobile-dev/native-vs-hybrid-which-mobile-app-platform-should-you-choose/> (acesso em 30/10/2022).

[37] *NativeBase: Universal components for React React Native*. URL: <https://nativebase.io/> (acesso em 08/02/2023).

[38] *Nota de repúdio*. URL: <http://www.fe.unb.br/index.php/noticia/527-nota-de-repudio> (acesso em 22/09/2022).

[39] *npm - Build amazing things*. URL: <https://www.npmjs.com/> (acesso em 03/02/2023).

[40] *O Slack é sua sede digital*. URL: <https://slack.com/intl/pt-br> (acesso em 11/02/2023).

[41] *Organização Mundial da Saúde*. URL: <https://www.who.int/pt> (acesso em 21/09/2022).

[42] *ORM, Query Builder or Raw SQL*. URL: <https://lyz-code.github.io/blue-book/architecture/orm_builder_query_or_raw_sql/> (acesso em 04/11/2022).

[43] *Os efeitos da violência*. URL: <https://www.pucrs.br/revista/os-efeitos-da-violencia/> (acesso em 21/09/2022).

[44] Victoria Pimentel e Bradford G Nickerson. “Communicating and displaying real-time data with websocket”. Em: *IEEE Internet Computing* 16.4 (2012), pp. 45–53.

[45] *Plans pricing - Life360*. URL: <https://www.life360.com/intl/intl-plans-pricing/> (acesso em 30/01/2023).

[46] *PostgreSQL: The world's most advanced open source relational database*. URL: <https://www.postgresql.org/> (acesso em 08/02/2023).

[47] Ely Fernando do PRADO. “Análise teórica sobre o desenvolvimento de aplicativos nativos, hıbridos e webapps.” Em: (2019).

[48] *Prisma | Next-generation Node.js and TypeScript ORM*. URL: <https://www.prisma.io/> (acesso em 04/11/2022).

[49] *React Native*. URL: <https://reactnative.dev/> (acesso em 30/10/2022).

[50] *Relacionamento muitos para muitos*. URL: <https://fmhelp.filemaker.com/help/18/fmp/pt/index.html#page/FMP_Help/many-to-many-relationships.html> (acesso em 03/02/2023).

[51] *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. URL: <https://www.rfc-editor.org/rfc/rfc2616> (acesso em 23/09/2022).

[52] *RFC 6455: The WebSocket Protocol*. URL: <https://www.rfc-editor.org/rfc/rfc6455> (acesso em 23/09/2022).

[53] Jair Vargas dos Santos, Marco Antônio Silveira de Souza e Daniel Fernando Anderle. “Controlando Dispositivos em Tempo Real Através do WebSocket”. Em: *Tecnologias e Redes de Computadores: estudos aplicados* 88960 (2015), p. 206.

[54] *Socket.IO*. URL: <https://socket.io/> (acesso em 03/02/2023).

[55] *socket.io - npm*. URL: <https://www.npmjs.com/package/socket.io> (acesso em 06/02/2023).

[56] *socket.io-client - npm*. URL: <https://www.npmjs.com/package/socket.io-client> (acesso em 06/02/2023).

[57] *Sou cidadão | Colab*. URL: <https://www.colab.re/> (acesso em 11/02/2023).

[58] *SQL Injection*. URL: <https://www.w3schools.com/sql/sql_injection.asp> (acesso em 04/11/2022).

[59] *SQL, NoSQL, NewSQL: Qual banco de dados usar?* URL: <https://blog.geekhunter.com.br/sql-nosql-newsql-qual-banco-de-dados-usar/#SQL_vs_NoSQL> (acesso em 04/11/2022).

[60] *SQLServer*. URL: <https://www.microsoft.com/pt-br/sql-server> (acesso em 08/02/2023).

[61] Nasima Ferdous Tripti et al. “SaveMe: a crime deterrent personal safety android app with a bluetooth connected hardware switch”. Em: *2018 9th IEEE Control and System Graduate Research Colloquium (ICSGRC)*. IEEE. 2018, pp. 23–26.

[62] *Uma visão geral do HTTP*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview> (acesso em 23/09/2022).

[63] *Using middleware*. URL: <https://expressjs.com/en/guide/using-middleware.html> (acesso em 01/02/2023).

[64] *Videoconferências, reuniões, chamadas | Microsoft Teams*. URL: <https://www.microsoft.com/pt-br/microsoft-teams/group-chat-software> (acesso em 11/02/2023).

[65] *Web App Vs Progressive Web App: Why Are They Different From Each Other?* URL: <https://www.csschopper.com/blog/web-app-vs-progressive-web-app/> (acesso em 30/10/2022).

[66] *WebSockets*. URL: <https://developer.mozilla.org/pt-BR/docs/Web/API/WebSockets_API> (acesso em 21/09/2022).

[67] *WebSockets Security: Main Attacks and Risks*. URL: <https://www.vaadata.com/blog/websockets-security-attacks-risks/> (acesso em 23/09/2022).

[68] *Welcome to Python.org*. URL: <https://www.python.org/> (acesso em 06/11/2022).

[69] *Welcome to Swift.org*. URL: <https://www.swift.org/> (acesso em 30/10/2022).

[70] *What Is a Push Notification? (+8 real-life examples)*. URL: <https://userguiding.com/blog/push-notifications/> (acesso em 01/02/2023).

[71] *WhatsApp*. URL: <https://www.whatsapp.com/?lang=pt_br> (acesso em 26/09/2022).

[72] *Why Prisma?* URL: <https://www.prisma.io/docs/concepts/overview/why-prisma> (acesso em 04/11/2022).

[73] *Xcode 14*. URL: <https://developer.apple.com/xcode/> (acesso em 06/02/2023).

[74] *Yarn - Package Manager*. URL: <https://yarnpkg.com/> (acesso em 03/02/2023).

[75] Ravi Sekhar Yarrabothu e Bramarambika Thota. "Abhaya: An Android App for the safety of women". Em: *2015 Annual IEEE India Conference (INDICON)*. IEEE. 2015, pp. 1–4.