UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTC - CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Nikolas Damian Borges Vasconcelos

Desenvolvimento de software de localização para alertas de áreas de risco

Florianópolis
2023

Nikolas Damian Borges Vasconcelos

Desenvolvimento de software de localização para alertas de áreas de risco

Trabalho de Conclusão de Curso do Curso de Graduação em Sistemas de Informação do CTC - Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Sistemas de Informação.
Orientador: Prof. Martin Augusto G. Vigil, Dr.

Florianópolis
2023

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.
Orientações em:
http://portalbu.ufsc.br/ficha

Nikolas Damian Borges Vasconcelos

Desenvolvimento de software de localização para alertas de áreas de risco

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação.

Florianópolis, 11 de Julho de 2023.

Prof. Álvaro Junio Pereira Franco, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Martin Augusto G. Vigil, Dr.
Orientador

Sidney Garcia B. Ferreira
Bacharel em Sistemas de Informação
Avaliador(a)
Unijui - Universidade Regional do Noroeste do Estado do Rio Grande do Sul

---

Prof.(a) Thaís Bardini Idalino, Dr(a).
Avaliador(a)
UFSC - Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço primeiramente a Deus pela vida, saúde, e oportunidade de poder concluir este curso apesar dos impasses. Sou grato também à minha esposa, que me deu todo suporte e motivação necessários para a conclusão deste projeto. Agradeço aos meus pais, irmão, madrasta, e aos meus colegas de curso que me acompanharam durante a jornada na Universidade Federal de Santa Catarina. Agradeço também aos meus grandes amigos Sidney Ferreira e Joelder Arcaro por me guieram em minha carreira profissional.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

RESUMO

Muitas vezes ocorrem eventos climáticos, como tempestades, vendavais, furacões, tornados, que trazem consequências trágicas para a população, sejam perdas financeiras ou de familiares. Estes eventos podem ser periódicos em algumas regiões, porém ocorrer raramente em outros, e as pessoas têm interesse de serem notificadas caso seja prevista alguma situação climática atípica, sejam estas moradoras do local específico ou apenas turistas. Neste contexto, este trabalho propõem o desenvolvimento e implantação de um aplicativo mobile com o propósito de poder notificar os cidadãos sobre possíveis incidentes climáticos, sendo estas notificações para a pessoa que está no local e para os familiares e pessoas conhecidas previamente autorizadas, sendo estas notificações enviadas baseadas na localização geográfica obtida pelo smartphone do usuário.

Palavras-chave: Aplicativo Mobile, React Native, Aplicativo Híbrido, Geolocalização, Segurança, Push Notification, Firebase.

ABSTRACT

Often climatic events occur, like storms, gales, hurricanes, tornadoes, which bring tragic consequences to the population, whether financial or family losses. These events may be periodic in some regions, but occur rarely in others, and people have an interest in being notified if an unusual climatic situation is anticipated, whether they are locals or just tourists. In this context, this work proposes the development and implementation of a mobile application with the purpose of being able to notify the citizens about possible climatic incidents, being these notifications for the person who is in the place and for the familiar and previously known persons known, being these notifications based on the geographic location obtained by the user's smartphone. With this goal in mind, we intend to evaluate the application with the help of volunteer users.

Keywords: Mobile App, React Native, Hybdri App, Geolocation, Security, Push Notification, Firebase.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
FCM	Firebase Cloud Messaging
FGV	Fundação Getúlio Vargas
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
INE	Universidade Federal de Santa Catarina
IOS	iPhone Operating System
NoSQL	No SQL
SMS	Short Message Service
SQL	Structured Query Language
TCC	Trabalho de conclusão de curso
UFSC	Universidade Federal de Santa Catarina
UI	User Interface
UX	User Experience

SUMÁRIO

1 INTRODUÇÃO

Seca, inundações, ciclones, enchentes. Com o aumento do aquecimento global provocado por atividades humanas, os desastres climáticos estão mais frequentes e intensos a cada ano que passa. A Organização das Nações Unidas alerta que a tendência é que esta situação se agrave com o decorrer do tempo (SCHUCK, 2021).

Com o crescente avanço da tecnologia, o Brasil vem se equiparado a países de primeiro mundo em relação ao uso de *smartphones*. Estima-se no Brasil hajam mais *smartphones* do que habitantes . Segundo a Fundação Getúlio Vargas (FGV), no país já existem cerca de 249 milhões de aparelhos (SOUZA, 2023).

Partindo deste ponto, é encontrada a necessidade de desenvolver uma solução para auxiliar no zelo pela segurança da população. O usuário que reside em áreas de risco poderá ser notificado em seu *smartphone* sobre previsões climáticas adversas, assim como turistas que estiverem passando por estas áreas, e familiares autorizados através do aplicativo também poderão ser notificados para poderem tomar as medidas desejadas.

Sempre que ocorre uma situação climática grave em alguma região do estado de Santa Catarina, a defesa civil envia mensagens de texto às pessoas avisando sobre a situação ocorrida (CIVIL, 2022). Neste caso, apenas as pessoas cadastradas ficam sabendo da situação, mas muitas vezes os parentes e entes queridos das pessoas que estão próximas à região do ocorrido ficam sabendo apenas no dia seguinte ou ainda mais tarde.

Atualmente o Facebook possui uma *feature* (META PLATFORMS, 2022) que visa atender esta necessidade, porém devido aos diversos escândalos em que a plataforma já se envolveu desde o início de sua existência (EDSON KAIQUE LIMA, 2021), muitas pessoas podem não se sentir à vontade compartilhando seus dados de localização com a plataforma constantemente.

Já o sistema utilizado pela Defesa Civil para a comunicação de eventos climáticos, ou no caso da pandemia utilizado para notificar sobre pessoas doentes na região, se baseia exclusivamente no envio de SMS, e não só pode haver uma certa demora natural na transferência de informações do ponto origem do ocorrido até as pessoas responsáveis pelo envio das mensagens, como são notificadas somente pessoas no local, e cabe a estas pessoas notificarem seus familiares.

Há também soluções similares nos Estados Unidos e que também podem ser usados no Brasil. De acordo com o blog (APPMASTER, 2022), existem diversos aplicativos móveis disponíveis tanto para iOS quanto para Android voltados para a temática abordada neste trabalho, porém eles se dividem entre aplicativos climáticos, aplicativos de alertas de emergência, e aplicativos de compartilhamento de localização familiar, mas nenhuma destas soluções une todas as funcionalidades em um único sistema, e nem que seja de uso dedicado e focado no público e nas necessidades do Brasil.

Tendo em vista esta situação, vem a ideia de desenvolver uma aplicação para

dispositivos móveis que notifique o usuário de possíveis situações climáticas inesperadas, e também notificar seus parentes e entes queridos cadastrados por ele na aplicação para que fiquem cientes do ocorrido no tempo devido.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver uma aplicação para dispositivos móveis que envia alertas de eventos climáticos ao usuário baseado em sua localização geográfica atual.

1.1.2 Objetivos Específicos

- Realizar um estudo comparativo de tecnologias de mercado para desenvolvimento mobile.
- Modelar e implementar um aplicativo mobile para alertas climáticos baseado na localização geográfica do usuário.

1.1.3 Escopo do trabalho

O presente trabalho tem por objetivo implementar um *software* para dispositivos móveis que seja uma alternativa para a atual solução fornecida pela Defesa Civil e pelo Facebook, analisando as ferramentas mais utilizadas no mercado e utilizando as que mais se adequarem às necessidades do projeto.

1.2 MÉTODO DE PESQUISA

Este trabalho pode ser categorizado como uma pesquisa aplicada. Ele será desenvolvido em 4 etapas:

1.2.1 Fundamentação teórica

Nesta etapa será analisada a fundamentação teórica, buscando informações de fontes fidedignas que abordem os assuntos e as tecnologias correspondentes com o tema tratado neste trabalho.

1.2.2 Projeto de Solução

Nesta etapa será elaborada uma proposta de solução baseada nos estudos e pesquisas realizadas nas etapas anteriores, bem como as tecnologias escolhidas para a implementação do aplicativo.

1.2.3 Desenvolvimento da Aplicação

Neste ponto do trabalho será detalhado como cada tecnologia foi utilizada, detalhes dos trechos de código mais cruciais utilizados para o funcionamento adequado do aplicativo, como as telas serão implementadas, bem como uma descrição da navegação pelo aplicativo realizada pelo usuário.

1.2.4 Avaliação das Funcionalidades

Neste último capítulo deste trabalho é apresentada uma avaliação das funcionalidades desenvolvidas no capítulo anterior, bem como demonstrações das mesmas.

1.3 ESTRUTURA DESTE DOCUMENTO

O Capítulo 2 deste trabalho abordará a fundamentação teórica do mesmo, tratando seus conceitos fundamentais e tecnologias relacionadas ao desenvolvimento de aplicações para dispositivos móveis nativas e híbridas. Serão abordadas informações sobre experiência de usuário, performance e consumo de bateria.

O Capítulo 3 abordará uma proposta de solução para o problema apresentado por este trabalho, utilizando-se das decisões técnicas tomadas no Capítulo 3.

No Capítulo 4 é apresentado o desenvolvimento do aplicativo, através das tecnologias detalhadas e apresentadas no Capítulo 4.

E por fim, no Capítulo 5 são apresentados os resultados obtidos a partir da implementação apresentada no Capítulo 4 ser finalizada. Neste capítulo é descrito como cada funcionalidade opera no dispositivo, como também resultados visuais da aplicação operando em dispositivo físico e virtual.

No Capítulo 6 são apresentadas as conclusões obtidas a partir do desenvolvimento da proposta, bem como situações que ocorrem durante a implementação do aplicativo que nem sempre são abordadas pela literatura.

E por fim, no Capítulo 7 são apresentados sugestões de trabalhos futuros para uma melhor utilização da aplicação desenvolvida.

2 FUNDAMENTAÇÃO TEÓRICA

Os avanços tecnológicos na área de dispositivos móveis têm revolucionado a forma como as pessoas interagem com a tecnologia. Com o crescimento exponencial do uso de *smartphones* e tablets, tornou-se essencial o desenvolvimento de aplicativos móveis que atendam às necessidades dos usuários de forma eficiente e intuitiva.

Esta sessão tem como objetivo apresentar os conceittos sobre tecnologias relacionadas ao desenvolvimento de aplicações para dispositivos móveis, abordando as principais tecnologias utilizadas em aplicações nativas e aplicações híbridas, dentro outros elementos-chave no desenvolvimento destes aplicativos.

2.1 APLICATIVOS MÓVEIS

De acordo com (KUBBEN, 2019), em 2018 iOS e Android juntos ocupavam aproximadamente 99% do mercado estadunidense de *smartphones*, sendo iOS 54% e Android 45%. O autor também menciona que há diferentes abordagens para o desenvolvimento de cada um, pois como será abordado a seguir, cada um possui uma forma diferente de ser desenvolvido e publicado.

2.1.1 Aplicações Nativas

As aplicações nativas são desenvolvidas especificamente para uma determinada plataforma, como Android ou iOS. Ainda também (KUBBEN, 2019) menciona que elas são escritas em linguagens de programação nativas, como Java ou Kotlin para Android e Objective-C ou Swift para iOS. Kuben também cita que essas aplicações têm acesso total aos recursos do dispositivo, como câmera, GPS e sensores, o que permite criar experiências ricas e personalizadas.

As vantagens das aplicações nativas incluem o desempenho otimizado, a possibilidade de aproveitar todos os recursos do dispositivo e a integração total com o sistema operacional. No entanto, o desenvolvimento de aplicações nativas requer conhecimentos específicos para cada plataforma, o que pode aumentar o tempo e o custo de desenvolvimento.

Porém graças à presença de ferramentas criadas para aplicações híbridas, o desenvolvimento para múltiplas plataformas tornou-se ainda mais atrativo aos olhos de empresas e desenvolvedores.

2.1.2 Aplicações Híbridas

Uma das principais vantagens das aplicações híbridas é a possibilidade de desenvolver uma única versão do aplicativo que funcione em várias plataformas, reduzindo o tempo e o custo de desenvolvimento. No entanto, as aplicações híbridas podem apresen-

tar um desempenho ligeiramente inferior em relação às aplicações nativas e podem ter acesso limitado aos recursos do dispositivo, estando à mercê de implementações/recursos implementados por empresas terceiras ou pela comunidade atuante na tecnologia de escolha.

O desenvolvimento de tecnologias híbridas destaca-se por utilizar dos artefatos *web*, como HTML, CSS e JavaScript, e são encapsuladas em um *container* nativo. Estes aplicativos podem ser executados em diferentes plataformas, como Android e iOS, a partir de um único código-base. Existem tecnologias populares para desenvolvimento de aplicações híbridas, como Apache Cordova, React Native e Flutter.

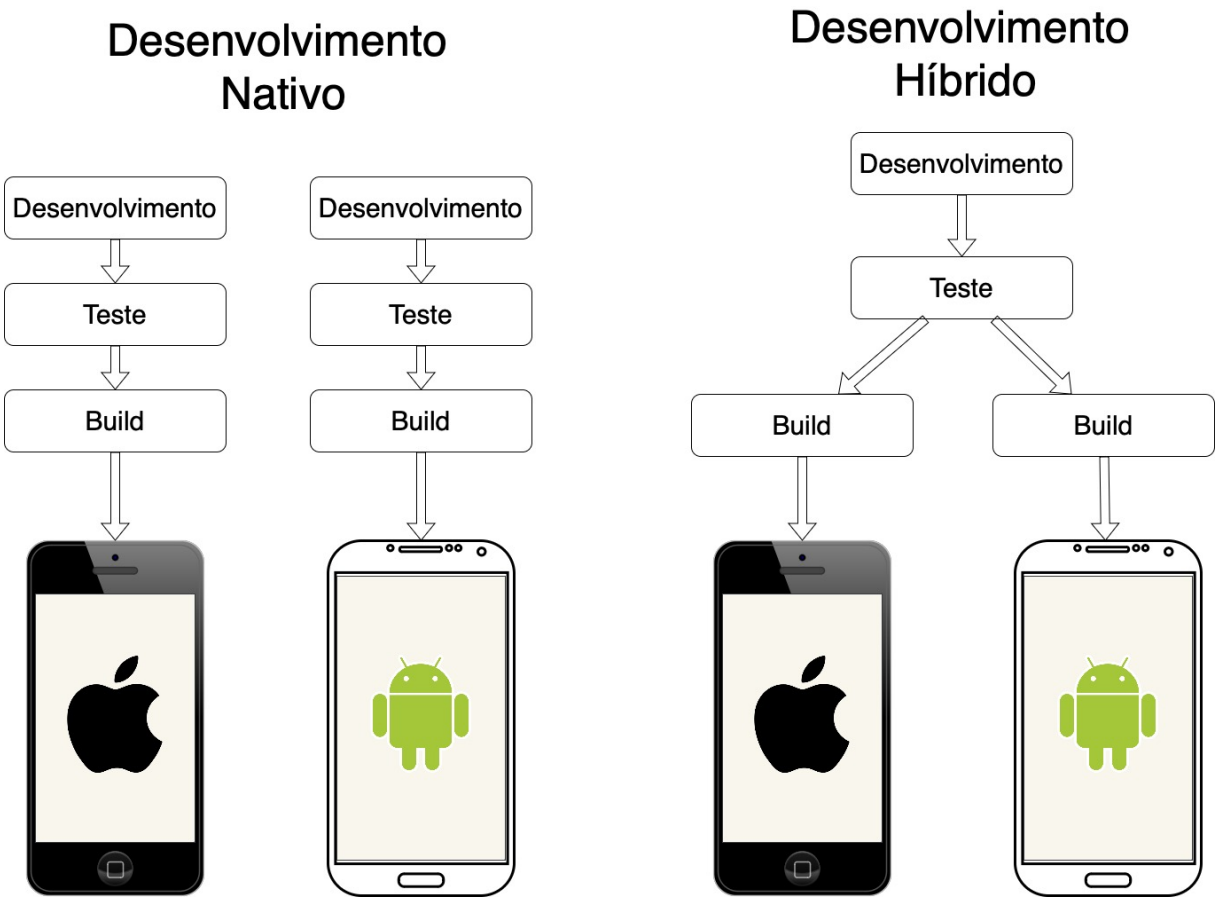


Figura 1 – Comparativo entre os estágios de desenvolvimento

Na Figura 2 é possível visualizar as principais diferenças entre o desenvolvimento de aplicativos móveis de forma nativa e como uma aplicação híbrida. Percebe-se que se torna mais prático e menos custoso desenvolver a aplicação somente uma vez e compilá-la para cada plataforma, ao invés de repetir o processo por completo a cada vez que os responsáveis pelo produto desenvolvido desejem publicar uma versão para uma plataforma diferente. Abaixo são apresentados mais detalhes sobre as tecnologias utilizadas no desenvolvimento de aplicativos híbridos.

2.1.2.1 HTML (HyperText Markup Language)

O HTML é a linguagem de marcação padrão para a criação de páginas web. Ele fornece a estrutura básica e os elementos que compõem o conteúdo de uma página, como texto, imagens, links e formulários. O HTML utiliza uma sintaxe simples e baseada em tags para definir a estrutura e a semântica do conteúdo.

O principal objetivo do HTML é fornecer uma estrutura lógica e hierárquica para os elementos de uma página web, permitindo que os navegadores interpretem e exibam o conteúdo corretamente. Com o HTML, os desenvolvedores podem criar páginas web acessíveis, organizadas e bem estruturadas.

2.1.2.2 CSS (Cascading Style Sheets)

O CSS é uma linguagem de estilo utilizada para definir a aparência e o layout dos elementos HTML em uma página web. Com o CSS, é possível controlar cores, fontes, espaçamentos, tamanhos e posicionamentos dos elementos, proporcionando uma apresentação visual agradável e consistente.

O CSS opera por meio de seletores, que identificam os elementos HTML a serem estilizados, e declarações, que especificam as propriedades de estilo a serem aplicadas. Além disso, o CSS permite a definição de regras de estilo em cascata, ou seja, a capacidade de aplicar estilos de forma hierárquica, permitindo a criação de estilos globais e específicos para determinados elementos.

2.1.2.3 JavaScript

O JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos. É amplamente utilizado no desenvolvimento web para criar interatividade, comportamento dinâmico e manipulação de elementos em uma página. Com o JavaScript, é possível responder a eventos, validar formulários, criar animações, fazer requisições assíncronas e muito mais.

O JavaScript é executado no lado do cliente, ou seja, diretamente no navegador do usuário. Ele permite a manipulação e a interação com os elementos HTML e CSS de uma página, tornando-a mais dinâmica e responsiva. Além disso, o JavaScript também pode ser executado no lado do servidor, por meio de frameworks como o Node.js, ampliando ainda mais suas possibilidades de uso.

2.1.2.4 TypeScript

O TypeScript, por sua vez, é uma extensão do JavaScript que adiciona recursos de tipagem estática, tornando o código mais legível e fácil de manter. O TypeScript ajuda a reduzir erros comuns de programação e facilita a colaboração em equipe, além de oferecer uma ampla variedade de recursos para o desenvolvimento de aplicativos móveis.

o TypeScript, juntamente com o React Native, oferece uma experiência de desenvolvimento mais eficiente e poderosa para aplicativos móveis modernos.

2.1.3 Aplicações Nativas vs. Híbridas

Ahmad *et al.* (2018) buscam mensurar as diferenças de performance entre aplicações híbridas e aplicações nativas desenvolvidas para o sistema operacional Android. De acordo com os autores, a diferença no tempo de implementação entre a aplicação nativa e a híbrida não é muito grande, porém se for levado em consideração que tal aplicação foi somente desenvolvida para Android, não é necessário a existência de uma aplicação híbrida. Porém se fosse considerado também o desenvolvimento para o sistema operacional iOS, o tempo de desenvolvimento, bem como os custos, da aplicação nativa tende a dobrar, pois o aplicativo terá que ser refeito em outra linguagem, mostrando assim que é mais produtivo desenvolver uma aplicação híbrida quando o objetivo do time de desenvolvimento é a implantação para mais de uma plataforma.

Entretanto, o time aborda as diferenças de performance como foco do trabalho, e tal não pode ser desconsiderado. Os autores mencionam que as diferenças de desempenho são bastante notáveis, e como este fator interfere diretamente na diferença do usuário, pode acarretar em um desempenho indesejado e más avaliações nas respectivas lojas onde os aplicativos são publicados. Com isto em mente, a tecnologia a ser utilizada e as abordagens durante o desenvolvimento devem ser muito bem pensadas e analizadas antes do início do projeto.

Gunawardhana (2021) apresenta diferentes resultados nos testes e análises de performance. Em seus gráficos comparativos, Gunawardhana mostra que, tanto em análises de experiências de usuário quanto em segurança e consumo de bateria, as aplicações desenvolvidas com tecnologia de aplicações híbridas obtiveram resultados similares ou superiores às aplicações nativas. Abaixo é apresentado o gráfico comparativo de performance entre um aplicativo Android híbrido, um nativo e um web, comparando seus usos de memória, tempo de processamento e segurança.

Com isto, pode-se observar que em comparação a (AJAYI *et al.*, 2018), com o passar do tempo as tecnologias utilizadas para o desenvolvimento híbrido tiveram um considerável avanço, sendo principalmente nos dias de hoje muito mais vantajosas em comparação a desenvolver uma aplicação nativa para cada plataforma desejada, mesmo se esta for apenas Android ou iOS.

2.2 ACESSO À LOCALIZAÇÃO EM TEMPO REAL

O acesso à localização em tempo real é uma das funcionalidades mais importantes dos aplicativos móveis modernos, permitindo que os usuários obtenham informações personalizadas com base em sua localização. Com a popularização dos smartphones e

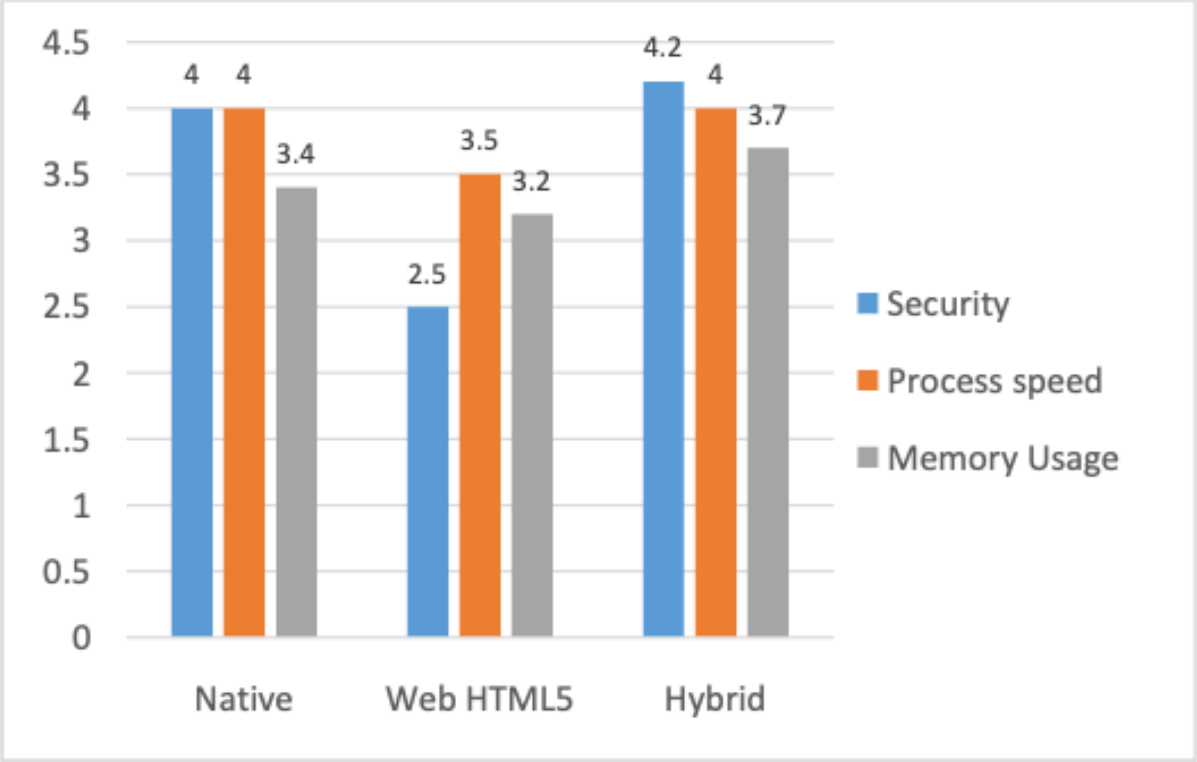


Figura 2 – Comparativo de performance entre Aplicativo Nativo, Web e Híbrido

a disponibilidade de GPS em praticamente todos os dispositivos, o acesso à localização tornou-se uma funcionalidade básica em muitos aplicativos móveis.

O acesso à localização em tempo real pode ser usado para fornecer informações sobre locais próximos, ofertas especiais ou eventos, além de oferecer uma experiência de usuário mais personalizada e relevante. O React Native oferece recursos nativos para acessar a localização em tempo real em aplicativos móveis, o que torna essa funcionalidade fácil de implementar e usar.

2.3 FIREBASE

O *Firebase* é uma plataforma de desenvolvimento de aplicativos amplamente utilizada que oferece uma variedade de serviços para ajudar os desenvolvedores a criar, melhorar e expandir seus aplicativos móveis. Ele simplifica várias tarefas complexas, fornecendo uma estrutura completa de desenvolvimento de aplicativos baseada em nuvem. Abaixo são listadas algumas funcionalidades cujo desenvolvimento torna-se simplificado através da integração com o Firebase.

2.3.1 Autenticação de Usuário

De acordo com (FORONES, 2021), a implementação de uma boa autenticação previne certos tipos de ataques à aplicação. O Firebase Auth é um serviço de autenticação

fornecido pelo Firebase que permite aos desenvolvedores adicionar facilmente autenticação aos seus aplicativos. Além dos métodos tradicionais, como login por email e senha, o Firebase Auth também oferece a opção de autenticação usando números de telefone.

A autenticação com número de telefone do Firebase Auth é uma solução segura e conveniente. Permite que os usuários verifiquem suas identidades usando seus números de telefone, recebendo um código de verificação por SMS. Essa abordagem elimina a necessidade de lembrar senhas e oferece uma experiência de login rápida e sem complicações.

2.3.2 Banco de Dados em Nuvem

O Firebase Firestore Database é um serviço de banco de dados NoSQL fornecido pelo Firebase. Ele permite que os desenvolvedores armazenem e sincronizem dados de forma eficiente em tempo real, facilitando a criação de aplicativos colaborativos e escaláveis.

O Firestore é um banco de dados baseado em documentos que oferece uma estrutura flexível para armazenar dados. Ele organiza os dados em coleções e documentos, onde cada documento é uma estrutura de chave-valor. Essa abordagem facilita o armazenamento e a recuperação de informações, permitindo consultas rápidas e eficientes.

Além da simplicidade de uso, o Firestore oferece recursos poderosos. Ele permite que os desenvolvedores sincronizem automaticamente os dados entre os dispositivos dos usuários em tempo real, garantindo que as alterações sejam refletidas instantaneamente. Isso é particularmente útil para aplicativos colaborativos, como aplicativos de chat em tempo real ou plataformas de compartilhamento de documentos.

O Firestore também oferece suporte a consultas avançadas, permitindo que os desenvolvedores realizem pesquisas complexas e filtrem os dados com base em critérios específicos. Essa flexibilidade permite a criação de recursos personalizados, como filtros de pesquisa, classificações e paginadores.

Outra vantagem do Firestore é sua integração perfeita com outros serviços do Firebase. Os desenvolvedores podem combinar o Firestore com o Firebase Auth para controlar o acesso aos dados com base nas permissões do usuário autenticado. Além disso, o Firestore pode ser usado em conjunto com o Firebase Cloud Functions para criar lógica de negócios personalizada e automatizar tarefas do lado do servidor.

O Firestore é altamente escalável e confiável. Ele lida automaticamente com o dimensionamento e a replicação dos dados, garantindo um desempenho consistente, mesmo para aplicativos com grande número de usuários e dados. Além disso, o Firestore oferece segurança robusta, protegendo os dados por meio de regras de segurança personalizáveis e autenticação do Firebase.

Em resumo, o Firebase Firestore Database é uma solução poderosa para armazenamento e sincronização de dados em tempo real. Sua estrutura flexível, recursos avançados e integração perfeita com outros serviços do Firebase tornam-no uma escolha popular entre os desenvolvedores de aplicativos móveis que buscam escalabilidade, colaboração e

uma experiência de usuário em tempo real.

2.3.3 Notificações *Push*

O Firebase Cloud Messaging (FCM) é um serviço de mensagens em nuvem fornecido pelo Firebase que permite aos desenvolvedores enviar notificações push para seus aplicativos. As notificações push são mensagens que são entregues diretamente aos dispositivos dos usuários, mesmo quando o aplicativo não está em execução.

O FCM oferece uma solução poderosa para o envio de notificações push em tempo real. Os desenvolvedores podem segmentar usuários específicos ou grupos de usuários e enviar mensagens personalizadas. As notificações podem incluir texto, imagens e até mesmo ações interativas, aumentando o engajamento e a retenção dos usuários.

3 PROJETO DE SOLUÇÃO

Este capítulo tem como objetivo apresentar uma proposta de aplicativo mobile para iOS e Android como solução, utilizando a tecnologia React Native. O aplicativo visa fornecer aos usuários alertas de risco climático em tempo real, com base em sua localização, além de permitir o envio de mensagens de texto (SMS), e notificações push, para pessoas selecionadas pelo usuário. O banco de dados será implementado utilizando o Realtime Database do Firebase, que possibilita registrar o banco localmente e em nuvem, e o login será realizado através do número de telefone do usuário utilizando o Firebase Auth.

Este aplicativo oferecerá aos usuários uma solução prática e eficiente para receber alertas de risco climático em tempo real. A implementação utilizando React Native e o Firebase permitirá o desenvolvimento rápido e a integração com serviços robustos de geolocalização, autenticação, armazenamento de dados e envio de notificações.

3.1 CASO DE USO

Para melhor análise das necessidades da aplicação e de seus requisitos, foi elaborado um diagrama de caso de uso para melhor compreensão. A partir deste, foram identificados três atores externos, tendo dentre eles o principal como sendo o usuário do aplicativo, de onde é disparado o alerta de incidente climático. Abaixo, na Figura 3, temos o diagrama de caso de uso que representa o principal ator da aplicação, que é o usuário cujo aplicativo instalado dispara o alerta de incidente climático.

3.2 DIAGRAMA DE FLUXO

Também para melhor análise do projeto e das necessidades da aplicação, foi elaborado um diagrama que represente o fluxo o qual a aplicação deve seguir, tendo como resultado o diagrama apresentado na Figura 4. Neste diagrama é apresentado o fluxo completo da aplicação, desde a abertura do aplicativo no dispositivo até o envio da mensagem ao contato favoritado. Ao inicializar o aplicativo, é verificada a autenticação do usuário. Se é um primeiro acesso ou espirou a sessão do usuário, este precisa passar pelo processo de login. Se não, ou depois do login, o usuário é redirecionado para a tela principal do mapa onde é possível visualizar a sua localização. Nesta tela, ele pode marcar os seus contatos desejados como favoritos, e/ou enviar um alerta aos contatos caso haja algum favoritado, e então é verificado se o contato favoritado é também um usuário do aplicativo ou não pra enviar a mensagem ou a notificação.

3.3 ARQUITETURA DA APLICAÇÃO

Após analisar os diagramas anteriores e as necessidades identificadas até então no projeto, bem como as tecnologias apresentadas anteriormente para o mesmo, foi elaborado

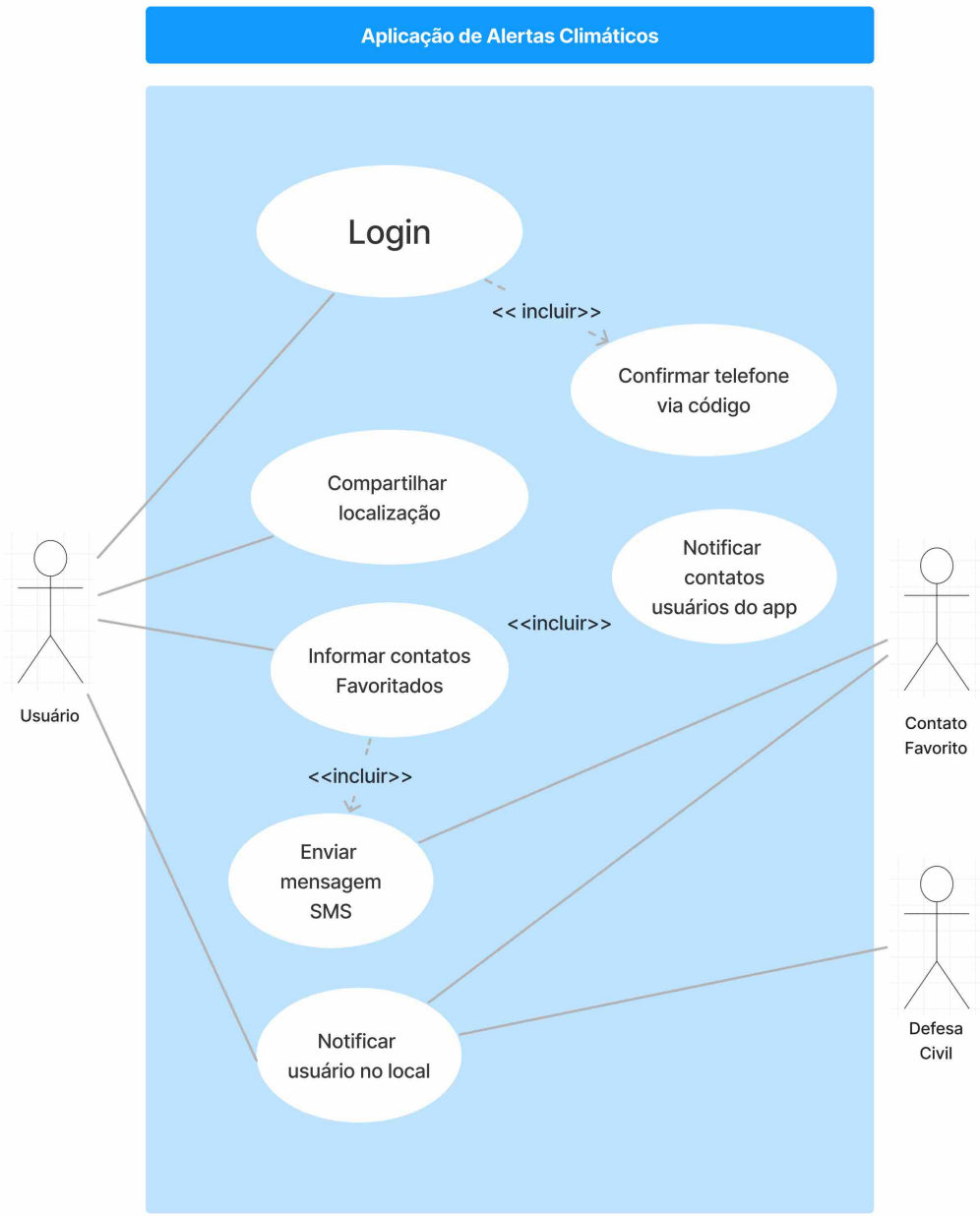


Figura 3 – Diagrama de Casos de Uso

o diagrama de arquitetura da aplicação demonstrado na Figura 5 abaixo. Pode-se perceber o elemento central, que é a aplicação do usuário, que se conecta com dois principais conjuntos de serviços externos: serviços referentes à plataforma *Google Cloud* (LLC, 2022), e serviços implementados que utilizam a localização do usuário para fornecer informações. Tais serviços se comunicam com a aplicação e também entre si, conforme demonstrado nas setas do diagrama.

3.4 MAPA COM ACESSO À LOCALIZAÇÃO EM TEMPO REAL

O aplicativo utilizará a API de localização do dispositivo para rastrear a posição do usuário em tempo real, utilizando a biblioteca *react-native-geolocation-service* (RIFAT, 2022). Será exibido um mapa interativo, através da biblioteca *react-native-maps* (AIRBNB, 2022), que mostrará a localização atual do usuário, bem como alertas de risco climático próximos à sua posição.

3.5 ENVIO DE MENSAGENS

O aplicativo permitirá que o usuário envie mensagens para pessoas selecionadas. O usuário poderá escolher pessoas de sua lista de contatos do dispositivo e enviar alertas de risco ou informações importantes para eles, através de SMS.

Para o envio de SMS será utilizada a feature de *Linking*, já inclusa na biblioteca *React Native.*. Esta feature permite acessar ao sistema de envio de SMS do próprio usuário (estando este sujeito às taxas de envio de seu provedor), sendo desta forma uma solução simples, prática e gratuita do ponto de vista de desenvolvimento, porém estando sujeita à confirmação do usuário para o envio da(s) mensagem(ns).

3.6 BANCO DE DADOS NA NUVEM E LOCAL

O *Firebase* será utilizado como plataforma de banco de dados para armazenar informações relevantes para o funcionamento do aplicativo. O Firebase possui o Realtime Database, que é um banco de dados NoSQL, e permite manter informações referentes à estrutura de dados da aplicação localmente no formato JSON, e sincronizá-las com o banco na nuvem assim que for houver estabelecido conexão. A ferramenta também possibilita a implementação de regras de segurança de acesso ao banco, como também a realização de backups.

3.7 LOGIN COM NÚMERO DE TELEFONE USANDO O *FIREBASE*

Para autenticação e login dos usuários, será utilizado o *Firebase Authentication* com a opção de login através do número de telefone. Isso permitirá que os usuários acessem o aplicativo utilizando seu número de telefone, aumentando a segurança e facilitando o processo de login.

3.8 ALERTAS DE RISCO CLIMÁTICO BASEADO NA LOCALIZAÇÃO DO USUÁRIO

Utilizando a API *Open Meteo* (OPEN-METEO, 2022), que fornece informações meteorológicas em tempo real , o aplicativo será capaz de alertar os usuários sobre possíveis riscos em sua localização atual. Serão exibidos alertas e notificações para que o usuário

possa estar ciente de condições meteorológicas perigosas e tomar as medidas apropriadas para sua segurança.

3.9 NOTIFICAÇÕES PUSH

Será implementada a funcionalidade de envio de notificações *push* para os usuários do aplicativo. Utilizando o *Firebase Cloud Messaging (FCM)*, será possível enviar notificações instantâneas para alertar os usuários sobre riscos climáticos iminentes ou informações relevantes.

As notificações poderão ser enviadas tanto a partir de um alerta identificado pelo app do usuário, notificando assim os demais contatos desejados que tiverem o app também instalado, ou por uma plataforma web caso seja implementada como um trabalho futuro.

3.10 CONTATOS

A lista de contatos do usuário será uma a exata lista do dispositivo, porém não será enviada à nuvem ou armazenada pela aplicação, e nem compartilhada ou tratada de alguma forma pela aplicação. O aplicativo apenas terá acesso de leitura dos contatos, e ao notificar, busca pelo número lido na lista de usuários.

3.11 LOCALIZAÇÃO

A aplicação não armazenará a localização do usuário em momento algum, não a utilizará para nenhum fim que não seja mostrar ao mesmo sua localização no mapa renderizado em tela, e compartilhará tal informação somente com o contato que o próprio usuário desejar enviar informações, podendo escolher no momento do compartilhamento da informação se lhe enviará a localização ou não. O compartilhamento com outros usuários ou contatos será através da *API* da *OpenCage* (GMBH, 2022), que permite realizar acessos gratuitos para testes, sendo utilizada para encontrar a descrição do local e poder enviar uma mensagem que seja compreensível ao usuário.


Usuário abre o App
Logado?
API Geolocalização
Não
Sim
Log In
Visualiza Localização
Banco de Dados de usuários
Não
Possui contatos Favoritados?
Sim
Não
Enviar alerta
Adicionar Favoritos
Pressiona botão de alerta
Contato é usuário do app?
Não
Sim
Envia SMS ao contato
Enviar notificação ao contato
Fim
Legenda
Início / Fim
Processo
Dados
Input

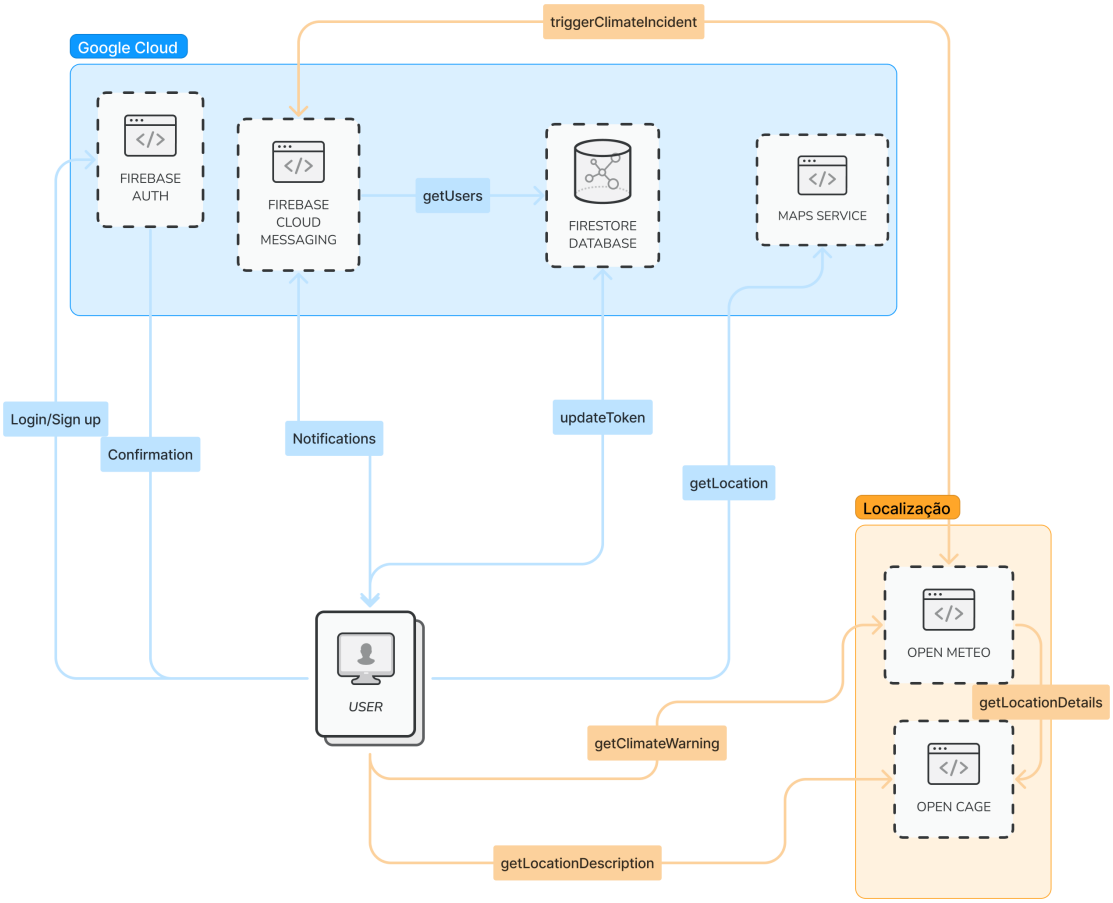


Figura 5 – Diagrama de Arquitetura

4 DESENVOLVIMENTO

Neste capítulo, será apresentada a metodologia adotada para o desenvolvimento e implantação do aplicativo para dispositivos móveis implementado utilizando a biblioteca React Native. Serão descritos os procedimentos e etapas realizadas ao longo do processo, a implementação do aplicativo, e a sua implantação. Cada uma das tecnologias utilizadas será abordada.

4.1 PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para o desenvolvimento de aplicações utilizando React Native, são necessários alguns passos para que tal seja possível. Em casos de desenvolvedores com pouco contato com a tecnologia, é recomendada a utilização da ferramenta *Expo* (EXPO, 2022), que é uma tecnologia construído em torno do React Native e possibilita ao desenvolvedor construir os aplicativos de forma mais rápida, bastando apenas ter uma versão recente do Node.js e de um telefone ou emulador da plataforma desejada.

4.1.1 React Native CLI v.s. Expo

Neste projeto foi utilizado o *React Native CLI* (METAOPENSOURCE, 2022), que requer as ferramentas Xcode e Android Studio, por ser possível trabalhar com as versões mais recentes disponíveis do React Native, possui melhor interoperabilidade, visto que em algumas etapas do desenvolvimento pode ser necessária configuração 100% nativa, produzir um artefato consideravelmente mais leve ao final do desenvolvimento, não são todas as APIs nativas que funcionam com a ferramenta *Expo*, como por exemplo a utilização com o Bluetooth, que pode ser utilizado futuramente neste projeto para uma integração com smartwatches, ou Android Auto/Apple CarPlay. Além da utilização do Xcode e Android Studio quando necessários, para o desenvolvimento principal da aplicação foi escolhido o editor de texto Visual Studio Code, que de acordo com a pesquisa feita por (OVERFLOW, 2022), é uma das ferramentas mais consolidadas do mercado para desenvolvimento de aplicações utilizando JavaScript ou TypeScript.

4.2 ESTRUTURA DO PROJETO

4.2.1 Inicialização do projeto

Por este projeto utilizar o React Native CLI, o comando utilizado para criação do projeto é o *npx react-native@latest init NomeDoProjeto*, que após executá-lo no terminal, cria um esqueleto do projeto já utilizando TypeScript por padrão, com todas as dependências iniciais necessárias tanto para Android quanto para iOS, e basta iniciar o desenvolvimento alterando o arquivo *App.tsx*, ou criando uma pasta dedicada para o

projeto como *src/* ou *app/* como uma aplicação web qualquer, desde que seguindo a sintaxe do React Native. Após o comando concluir sua execução, o projeto criado terá a estrutura de pastas apresentada na Figura 6:

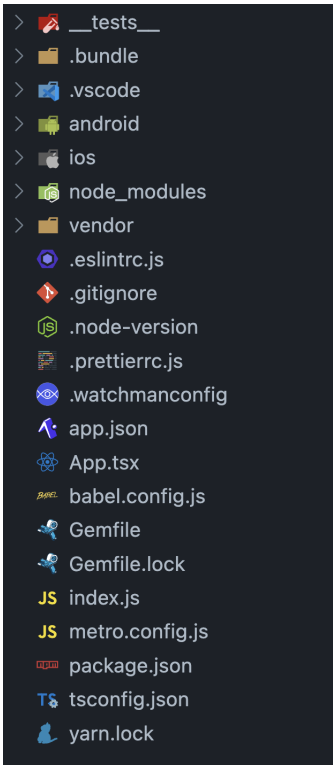


Figura 6 – Estrutura de pastas de um projeto React Native iniciado com o seu comando padrão

As pastas ios e android são as pastas correspondentes de cada plataforma a qual será destinada a aplicação, contendo nelas quaisquer configurações específicas necessárias para a implantação do aplicativo em sua respectiva plataforma. Caso o desenvolvedor deseje realizar alguma alteração específica utilizando os softwares Xcode ou Android Studio, basta executá-los e, ao escolher o projeto que deseja abrir através do software, selecionar a pasta ios ou android respectivamente.

A imagem apresentada na Figura 7 mostra a estrutura de pastas do projeto após o desenvolvimento dos componentes, telas, serviços e integrações. Todo o desenvolvimento da aplicação se encontra dentro da pasta *src/*.

4.3 PERMISSÕES

Para poder operar corretamente, a aplicação necessita de acesso a recursos nativos do dispositivo, como acesso à localização em tempo real, à lista de contatos, ao envio de SMS, e ao recebimento de notificações. Para isto é utilizada a biblioteca *react-native-permissions* (ACTHERNOENE, 2022), cuja função é poder solicitar ao usuário permissão

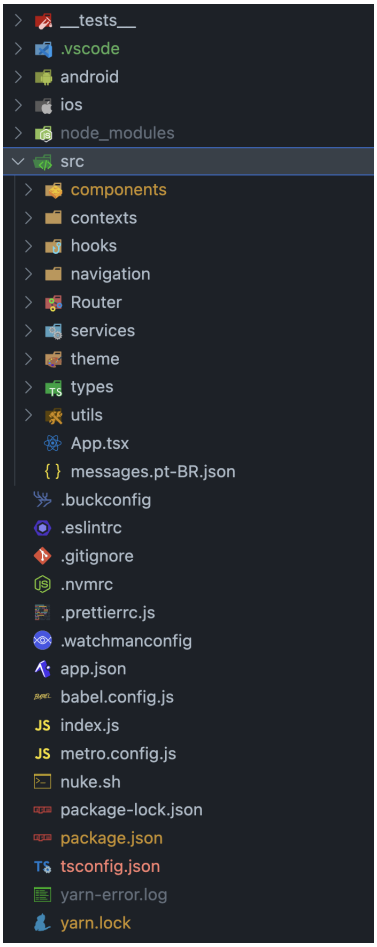


Figura 7 – Estrutura de pastas deste projeto

para acesso às funcionalidades que forem necessárias para o funcionamento correto da aplicação, e outros também como utilização da câmera, microfone, entre outros.

4.4 INTERFACE DA APLICAÇÃO

Nesta sessão serão apresentadas as telas desenvolvidas para esta aplicação, elencadas em ordem de acesso pelo usuário.

4.4.1 Autenticação

As telas abaixo demonstram o fluxo de autenticação do usuário na aplicação. Ao executar o aplicativo, ao usuário é apresentada a tela de Login, onde lhe é solicitado o seu número de telefone. Ao confirmar, segue para a tela de confirmação, onde ele deverá inserir o código de 6 dígitos que recebeu via SMS pelo sistema de autenticação da ferramenta Firebase Auth. O código estando correto, seu número de telefone é salvo no banco de dados Realtime Database (também do Firebase), seu token para recebimento de notificações é atualizado, e então ocorre o redirecionamento para a tela principal da aplicação.

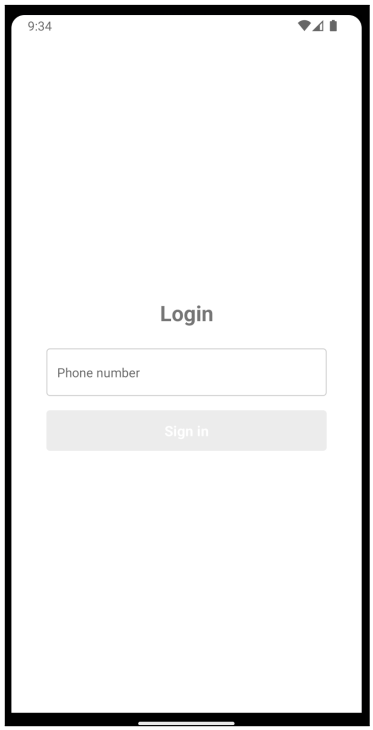


Figura 8 – Tela de Login

A tela apresentada na Figura 8 é responsável pelo login do usuário, utilizando seu número de telefone.

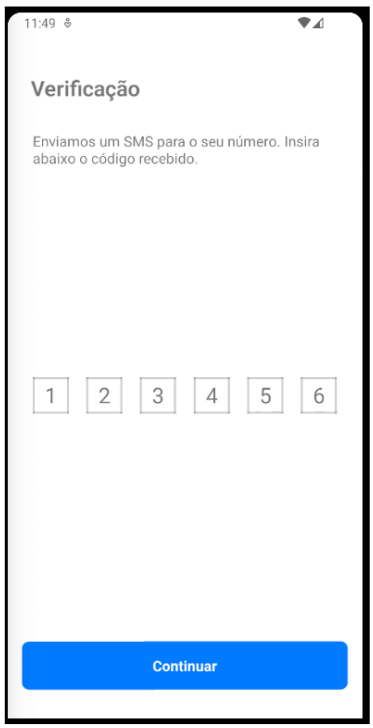


Figura 9 – Validação do código enviado por SMS

Na tela apresentada na Figura 9, o usuário insere o código recebido via SMS em

seu dispositivo após o Firebase identificar seu número de telefone.

4.4.2 Telas Pós Login

Abaixo, na Figura 10, é apresentada a tela principal da aplicação, onde o usuário pode visualizar sua localização em tempo real, interagir com o mapa utilizando suas funções básicas de zoom. . Na parte inferior do mapa há o sistema de navegação por abas, onde o usuário tem acesso às duas telas da aplicação: a tela atual do mapa, e à tela de contatos.

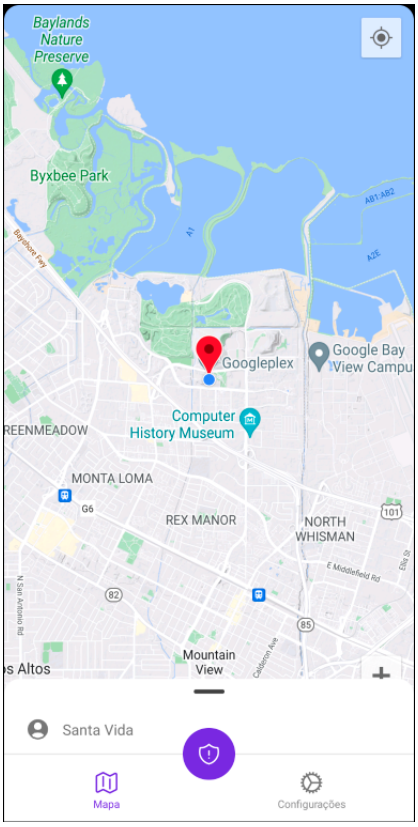


Figura 10 – Tela mapa

Juntamente com o mapa, logo acima da há um componente conhecido como *Bottom Sheet*, que funciona como uma gaveta, e pode ser arrastado para cima. Neste *Bottom Sheet* estão a lista de contatos favoritos do usuário.

Na parte central das abas, há um botão em destaque que é utilizado para envio de alertas manualmente. Caso o usuário esteja em alguma situação que não necessariamente seja uma questão climática, como por exemplo durante uma pandemia o usuário descubra que está infectado com o vírus, e teve contato recente com alguns de seus contatos ou deseja que eles saibam da situação. É possível enviar detalhes

A outra tela da aplicação é a tela de configurações, apresentada na Figura 11, onde o usuário pode ver todos os contatos do dispositivo do usuário, ou fazer *log out* da aplicação. Na tela de contatos, apresentada na Figura 12, ao lado de cada nome de contato

há uma estrela que funciona como um botão de *toggle*, e quando a estrela está com o fundo branco, é um contato desmarcado como favorito, e quando está com o fundo preenchido, o contato está marcado como favorito. Também é possível acessar as informações do contato na própria agenda do dispositivo ao pressionar o nome de um do contato desejado.

Figura 11 – Tela Configurações

4.5 INTEGRAÇÕES

Nesta sessão serão apresentadas as integrações realizadas na aplicação para a implementação de suas das funcionalidades principais.

4.5.1 Firebase

4.5.1.1 Recebimento de notificações

As notificações da aplicação funcionam a partir da biblioteca *@react-native-firebase/messaging* (LIMITED, 2022a), que funciona como uma interface de comunicação com os serviços nativos do *FCM (Firebase Cloud Messaging)*. Através dela é possível configurar o recebimento de notificações. Quando a aplicação inicia, um token é gerado e armazenado localmente no dispositivo, e este *token* é utilizado para identificar a notificação direcionada ao mesmo. Após o *token* gerado, um serviço de comunicação

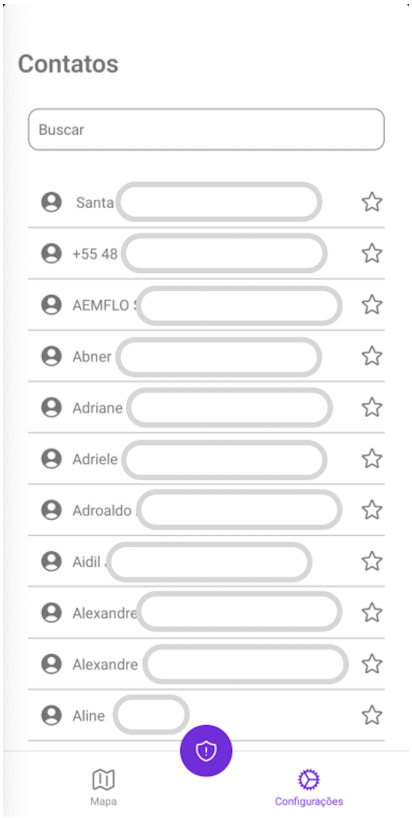


Figura 12 – Tela de contatos. Os sobrenomes foram cobertos por motivos de privacidade do desenvolvedor

com o *FCM* para identificar quando uma notificação foi enviada e mostrá-la ao usuário, conforme o código abaixo apresentado.

```
1
2  export async function handleTokenRefresh(): Promise<() => void> {
3    return messaging().onTokenRefresh((fcmToken) => {
4      if (fcmToken) {
5        updateFcmToken(fcmToken);
6      }
7    });
8  }
9
10 export async function updateFcmToken(token: string) {
11   await updateUser({ fcmToken: token });
12 }
13
14 export async function getFcmToken(): Promise<string | null> {
15   try {
16     const fcmToken = await messaging().getToken();
17     if (fcmToken) {
18       return fcmToken;
19     }
20   } catch (error) {
```

```
21      // Ignore err
22    }
23    return null;
24  }
```

Listing 4.1 – Métodos utilizados para geração do token do serviço

Após a geração e recebimento do token, o aplicativo pode identificar as notificações enviadas ao dispositivo onde está instalado, e assim realizar quaisquer tipos de tratamento de dados necessário, ou como no caso desta aplicação, simplesmente informar algo ao usuário, através dos códigos abaixo apresentados.

```
1
2   useEffect(() => {
3     if (notificationsAllowed) {
4       listenToBackgroundMessages();
5       listenToMessages();
6     } else {
7       requestPermission();
8     }
9   }, [notificationsAllowed]);
```

Listing 4.2 – Hook responsável pela identificação de notificações recebidas

```
1
2   export async function listenToMessages(): Promise<(() => void) |
      undefined> {
3     try {
4       const onMessageUnsub = await messaging().onMessage(async ({ data })
          => {
5         const currentMessages = await getPersistedItem(FCM_MESSAGES_KEY);
6         if (currentMessages) {
7           const messageArray = JSON.parse(currentMessages);
8           messageArray.push(data);
9           await setPersistentItem(
10            FCM_MESSAGES_KEY,
11            JSON.stringify(messageArray),
12          );
13        }
14      });
15      return onMessageUnsub;
16    } catch (error) {
17      // Ignore err
18    }
19  }
20
21  export async function listenToBackgroundMessages(): Promise<void> {
22    try {
23      await messaging().setBackgroundMessageHandler(async ({ data }) => {
```

```
24        const currentMessages = await getPersistedItem(FCM_MESSAGES_KEY);
25        if (currentMessages) {
26          const messageArray = JSON.parse(currentMessages);
27          messageArray.push(data);
28          await setPersistentItem(
29            FCM_MESSAGES_KEY,
30            JSON.stringify(messageArray),
31          );
32        }
33      });
34    } catch (error) {
35      // Ignore err
36    }
37  }
```

Listing 4.3 – Métodos responsáveis pelo tratamento das mensagens recebidas

4.5.1.2 Banco de dados

Através da biblioteca *@react-native-firebase/database* (LIMITED, 2022b), é possível realizar a integração com o Real Time Database, onde os números de telefone dos usuários serão registrados juntamente com seus tokens gerados descritos no tópico anterior. Através desde banco de dados, o aplicativo pode identificar quais de seus contatos marcados como favoritos também possuem o aplicativo instalado em seus dispositivos e serem notificados. Segue abaixo o código responsável por tais operações:

```
1
2  export const updateFirebaseDatabase = async (phoneNumber, fcmToken) => {
3    const databaseRef = firebase.database().ref('users');
4
5    const userId = firebase.auth().currentUser.uid;
6    await databaseRef.child(userId).set({
7      phoneNumber: phoneNumber,
8      fcmToken: fcmToken,
9    })
10 }
```

Listing 4.4 – Atualizar dados do usuário no banco de dados

Tendo o banco de dados atualizado com as informações necessárias, agora é necessário configurar as Cloud Functions do Firebase. Como o envio de notificações é realizado somente a partir do Console do Firebase, é necessário criar funções que farão o envio das notificações pelo Console ao invés de uma pessoa manualmente.

Primeiramente é criado a pasta ”functions” no diretório raiz do projeto, e dentro dela um arquivo ”index.ts”. Neste arquivo, é inserido o seguinte código:

```
1  const functions = require('firebase-functions');
```

```
2  const admin = require('firebase-admin');
3
4  admin.initializeApp();
5
6  const sendPushNotifications =
7    functions.https.onRequest(async (req, res) => {
8      const phoneNumbers = req.body.phoneNumbers;
9      const payload = {
10       notification: {
11         title: 'Alerta!',
12         body: `${req.body.name} lhe informa: ${req.body.message}`,
13       },
14     };
15
16     try {
17       const tokensSnapshot = await admin
18         .database()
19         .ref('users')
20         .once('value');
21       const tokens = [];
22
23       tokensSnapshot.forEach((childSnapshot) => {
24         const user = childSnapshot.val();
25         if (phoneNumbers.includes(user.phoneNumber) && user.fcmToken) {
26           tokens.push(user.fcmToken);
27         }
28       });
29
30       if (tokens.length > 0) {
31         await admin.messaging().sendToDevice(tokens, payload);
32         res.status(200).send('Notifica  es enviadas com sucesso');
33       } else {
34         const phones = phoneNumbers.join(',')
35         res.status(200)
36           .send(
37             `Estes contatos n o possuem o app instalado: ${phones}`
38           );
39       }
40     } catch (error) {
41       console.error('Error sending push notifications:', error);
42       res.status(500).send('Error sending push notifications.');
43     }
44   });
```

Listing 4.5 – Notificar múltiplos usuários simultaneamente

Neste código é verificado a existência dos tokens e, caso haja algum, a notificação é enviada para todos ao mesmo tempo.

Para que este método seja invocado no código, pasta realizar uma requisição POST para a url do projeto, chamando a API "sendPushNotifications", passando como corpo da requisição os números de telefone como um *array*, o nome do usuário que enviou as notificações e a mensagem gerada pela aplicação.

4.5.2 Alertas Climáticos

Através da API 100% gratuita e *Open Source* da plataforma Open Meteo (OPEN-METEO, 2022), é possível implementar o serviço de alertas climáticos. A ferramenta opera utilizando um sistema de códigos, os quais devem ser identificados a cada alerta. Os métodos responsáveis por tais implementações se encontram abaixo:

```
1  import axios from 'axios';
2
3  export async function fetchWeatherData(city) {
4    const apiUrl = `https://api.open-meteo.com/weather?location=${city}`;
5
6    try {
7      const response = await axios.get(apiUrl);
8      return response.data;
9    } catch (error) {
10     console.error('Error fetching weather data:', error);
11     throw error;
12   }
13 }
14
15 export function checkForWeatherDangers(weatherData) {
16   const weatherConditions = weatherData.weather.code;
17   const warnings = [];
18
19   if (weatherConditions === 5 || weatherConditions === 6) {
20     warnings.push('Perigo! Tempestade a caminho!');
21   }
22
23   if (weatherConditions === 9 || weatherConditions === 10) {
24     warnings.push('Perigo! Pode haver um tornado pr ximo a voc !');
25   }
26
27   if (weatherConditions === 16) {
28     warnings.push('Perigo! Inunda  o em sua regi o!');
29   }
30
31   if (weatherConditions === 25) {
32     warnings.push('Perigo! Chuva forte a caminho!');
33   }
34
35   if (weatherConditions === 26) {
```

```
36      warnings.push('Perigo! Uma nevasca se aproxima!');
37    }
38
39    return warnings;
40  }
```

Listing 4.6 – Métodos responsáveis por buscar informações climáticas baseados na localização do usuário

Juntamente com esta implementação, é possível identificar tais acontecimentos em segundo plano através da biblioteca *react-native-background-fetch* (SCOTT, 2022), que permite que a aplicação execute tarefas em segundo plano com a permissão do usuário. Desta forma, é possível enviar as notificações sem depender da interação do usuário.

4.5.3 Mensagens via SMS

As mensagens na aplicação poderão ser enviadas manualmente via SMS aos contatos favoritos através da feature de *Linking*, disponível nativamente pela biblioteca React Native, que permite à aplicação acessar à função nativa de envio de SMS do próprio dispositivo, utilizando o provedor do usuário. Por se tratar de utilização dos serviços de envio de SMS contratados pelo próprio usuário, a finalização desta ação depende da confirmação do mesmo, tornando assim o envio das mensagens opcional.

```
1  export function sendSMS(contacts: string[], message: string) {
2    Linking.openURL(`sms:${contacts}?body=${message}`)
3  }
```

Listing 4.7 – Método responsável pelo envio de SMS

No código acima, o método "*sendSMS*" recebe por parâmetro um vetor de números de telefone e a mensagem que deve ser enviada aos contatos. Através do método "*openURL*", que recebe uma *string* por parâmetro, e recebendo assim o valor "sms:" no início da *string*, o método invoca a função padrão de envio de SMS do dispositivo.

4.5.4 Localização

Através da *API* de busca por geolocalização da *OpenCage* (GMBH, 2022) é possível enviar uma mensagem mais completa ao contato desejado. Na plataforma da *OpenCage* (GMBH, 2022), cria-se uma conta gratuitamente. Lá a plataforma fornece uma chave de *API*, a qual é necessária para realizar as requisições *REST* para obter descrições sobre a localização através da latitude e longitude. No código abaixo, segue a *API* utilizada para obter a localização do usuário e a resposta recebida no formato *JSON*.

```
1
2  import axios from 'axios';
3  import { OPENCAGE_KEY } from '@env'
4
```

```
5  export async function getLocationDetails(latitude: number, longitude:
       number): Promise<string> {
6    const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude
       }+${longitude}&key=${OPENCAGE_KEY}&language=pt-BR`
7    console.log(`URL::`, url)
8
9    try {
10     const response = await axios.get(url);
11
12     if (response.data && response.data.results.length > 0) {
13       const result = response.data.results[0];
14       return result.formatted;
15     }
16
17     throw new Error('No results found');
18   } catch (error: any) {
19     console.error('Error retrieving location name:', error.message);
20     throw error;
21   }
22 }
23
24
25 export async function getCityName(latitude: number, longitude: number):
       Promise<{ city: string, state: string }> {
26   const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude
       }+${longitude}&key=${OPENCAGE_KEY}&language=pt-BR`
27   console.log(`URL::`, url)
28
29   try {
30     const response = await axios.get(url);
31
32     if (response.data && response.data.results.length > 0) {
33       const result = response.data.results[0];
34       const components = result.components;
35       const city = components.city || components.town || components.
           village;
36       const state = components.state;
37
38       if (city && state) {
39         return { city, state };
40       }
41     }
42
43     throw new Error('No results found');
44   } catch (error: any) {
45     console.error('Error retrieving location name:', error.message);
46     throw error;
```

```
47   }
48 }
```

Listing 4.8 – Método responsável pela busca de dados do clima na localização do usuário

```
{
 ...
 "results": [
 {
 ...
 "formatted": "Universidade Federal de Santa Catarina, Rua Roberto Sampaio Gonz
 ...
 }
 ],
 ...
}
```

Através da requisição da API de Geocoding da empresa *OpenCage* com o método *GET* , pode-se obter a descrição completa da localização do usuário pelo objeto *response.data.results[0].formatted*, que neste caso do código retorna a seguinte informação: "Universidade Federal de Santa Catarina, Rua Roberto Sampaio Gonzaga, Trindade, Florianópolis - SC, 88040-900, Brasil".

4.6 IMPLANTAÇÃO

Devido à intenção deste trabalho de ser uma aplicação reutilizada por instituições interessadas, como a Defesa Civil por exemplo, esta aplicação não será implantada. Em caso de desejo de implantação, esta pode ser feita através das plataformas *Google Play Console* (para dispositivos Android) e *App Store Connect* (para dispositivos iOS).

5 AVALIAÇÃO FUNCIONAL

Após o desenvolvimento da aplicação e suas funcionalidades, foram realizados testes das funcionalidades para confirmar a efetividade da aplicação. Para tal, foram usados um simulador de dispositivo Android, simulando um *smartphone* com sistema operacional Android 12, em um notebook modelo Macbook Pro 2023, e um smartphone com sistema operacional Android 13, onde o simulador dispara os eventos que devem ser escutados pelo smartphone.

5.1 LOGIN

Na sessão de login, a implementação realizada utilizando-se da integração da ferramenta *Firebase Auth* mostrou-se bastante simples e fluida conforme o esperado, sendo apresentado na Figura 13 e Figura 14:

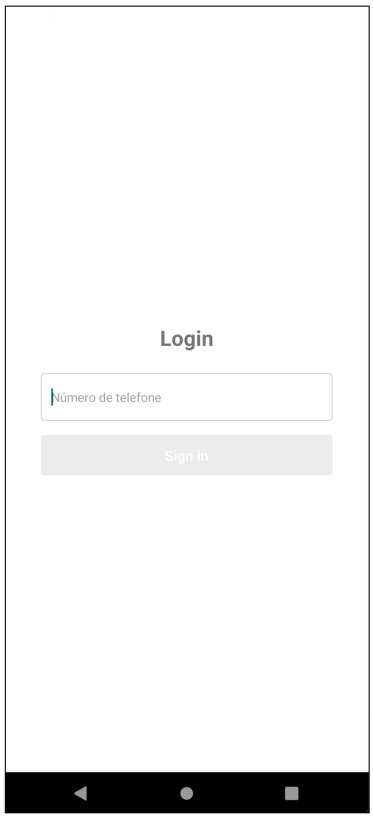


Figura 13 – Tela Login com campo vazio

Após o número de telefone inserido, a aplicação navega até a tela responsável por inserir o número de SMS recebido pelo telefone, como é demonstrado na Figura 15. Como esta é uma *build* de testes, é configurado no console do firebase um número de telefone e um código de verificação para que possam ser realizados testes sem que hajam possíveis custos com a plataforma. Nesta situação, foi cadastrado o número apresentado na Figura 14, e o código 123456 para testes, conforme apresentado na Figura 16:

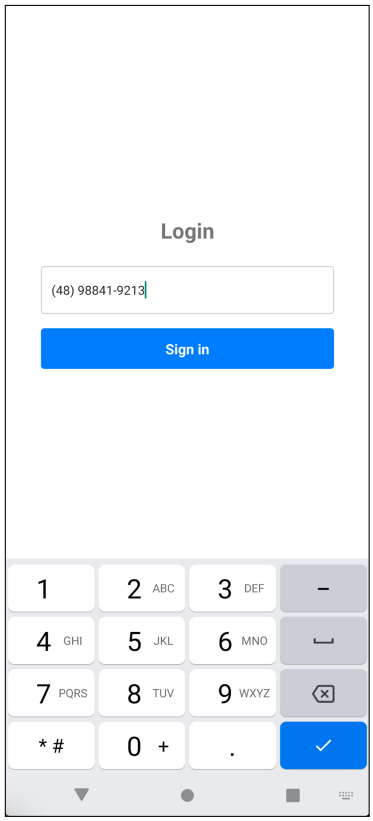


Figura 14 – Tela de login com campo preenchido

Conforme demonstrado na imagem acima, após a verificação do código inserido, o sistema automaticamente identifica o usuário logado e redireciona para as telas principais da aplicação.

5.2 ALERTAS ENVIADOS AOS CONTATOS

5.2.1 Alertas por Notificações Push

Através da utilização das *API*s da plataforma *Open Meteo* e *Open Cage*, foi possível implementar *listeners* para possíveis dificuldades climáticas e, estas sendo identificadas, notificar aos usuários marcados como favoritos que possuíam a aplicação instaladas em seus *smartphones* através do aplicativo instalado no simulador.

Para este teste, devido à ausência de "catástrofes"na região, foi necessário simular respostas aceitáveis para as situações onde as notificações seriam enviadas. A seguir são apresentados os resultados na tela do aplicativo nos dispositivos.

Na Figura 17 é possível ver a notificação recebida enquanto o aplicativo não estava aberto no *smartphone*, mostrando o nome do contato que enviou a notificação, bem como o incidente climático e o local de forma que seja possível uma simples compreensão.

Na Figura 18 a seguir, é possível visualizar o alerta advindo de outro contato quando o aplicativo está aberto no smartphone do usuário favoritado.

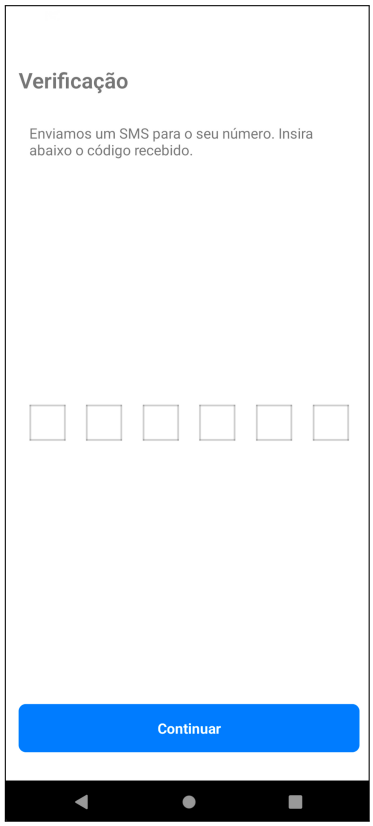


Figura 15 – Tela de confirmação via SMS com campos vazios

Figura 16 – Tela de confirmação via SMS com campos vazios

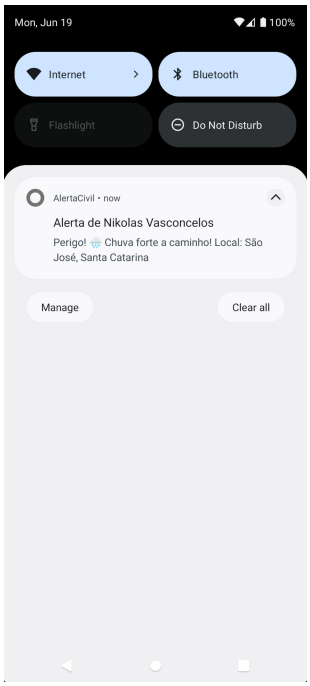


Figura 17 – Notificação em segundo plano

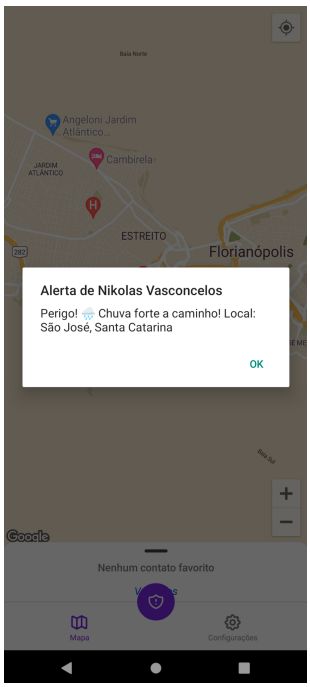


Figura 18 – Notificação com aplicativo aberto

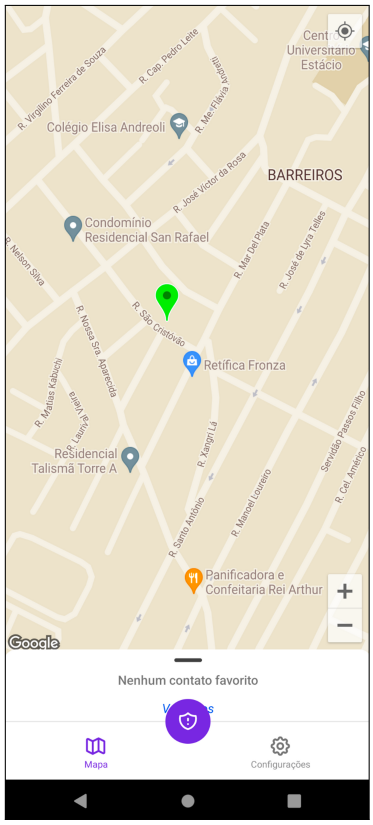


Figura 19 – Localização advinda da notificação recebida

Através da ferramenta Firebase Cloud Messaging é possível também enviar dados através das notificações push. Neste projeto foi utilizada esta funcionalidade para que fossem enviadas as coordenadas do incidente climático para o usuário favoritado. Desta forma, foi possível mover o mapa no aplicativo para o local origem da notificação, como demonstrado na Figura 19 abaixo no mapa do aplicativo o marcador destacado com a cor verde.

Na Figura 20 acima é possível visualizar a localização do incidente, em verde, e a localização do dispositivo que recebeu a localização através de notificação *push*.

5.3 ALERTAS VIA SMS

Através da funcionalidade nativa de *Linking* do React Native, ao pressionar o botão central de alerta, foi possível abrir o aplicativo padrão de envios de SMS do dispositivo, o qual automaticamente selecionou os contatos favoritos na agenda e adicionou a mensagem previamente configurada no input de mensagens. Utilizando-se também da *API* da plataforma *Open Cage* (GMBH, 2022) foi possível enviar as mensagens com detalhes da localização via SMS. Foi considerado necessário informações mais detalhadas no envio via SMS visto que nem todos os contatos marcados como favoritos receberiam tal notificações por não ter o aplicativo de alertas instalado.

A Figura 21 abaixo mostra os contatos marcados previamente como favoritos pelo

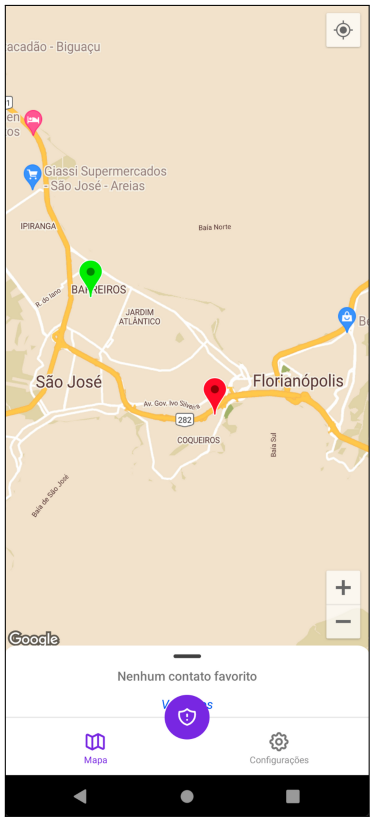


Figura 20 – Marcadores de localização do usuário e do contato

usuário na lista de contatos do aplicativo.

A Figura 22 abaixo demonstra a mensagem a ser enviada via SMS para os contatos favoritos simultaneamente.

A Figura 23 abaixo demonstra a mensagem já enviada para ambos contatos. Conforme o aplicativo de mensagens do *smartphone* informa, ambos os contatos foram comunicados de forma individual, e desta mesma forma responderão caso o decidam fazer.

O contato que possuía o aplicativo instalado também recebeu uma notificação *push* juntamente com a SMS enviada, conforme demonstrado na Figura 24 abaixo. A notificação nesta situação possui o exato mesmo conteúdo da SMS.

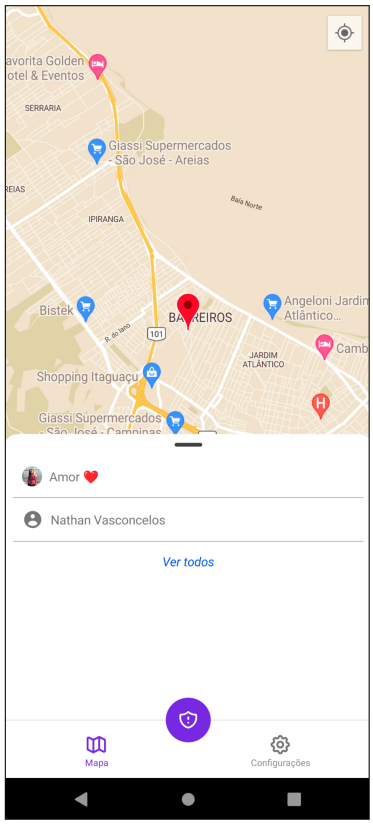


Figura 21 – Contatos marcados como favoritos

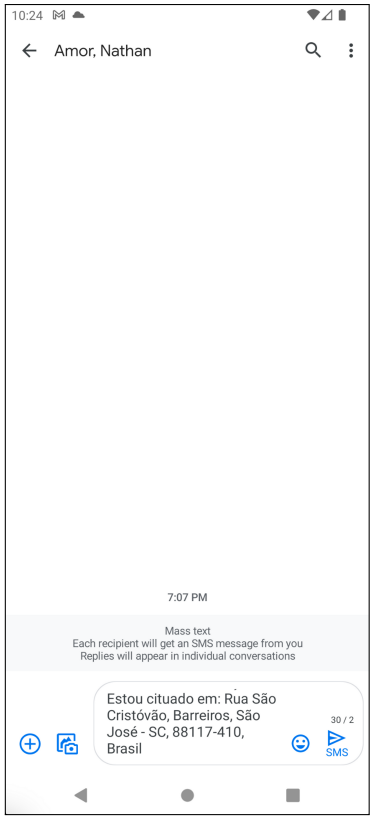


Figura 22 – Aplicativo de SMS aberto com a mensagem no input de texto

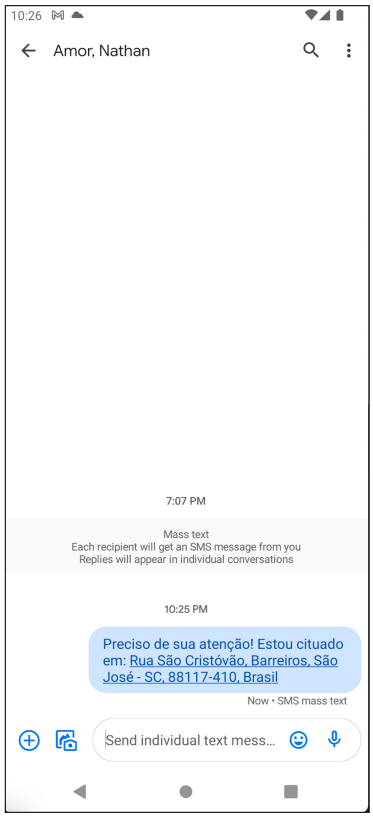


Figura 23 – Aplicativo de SMS aberto com a mensagem enviada aos contatos

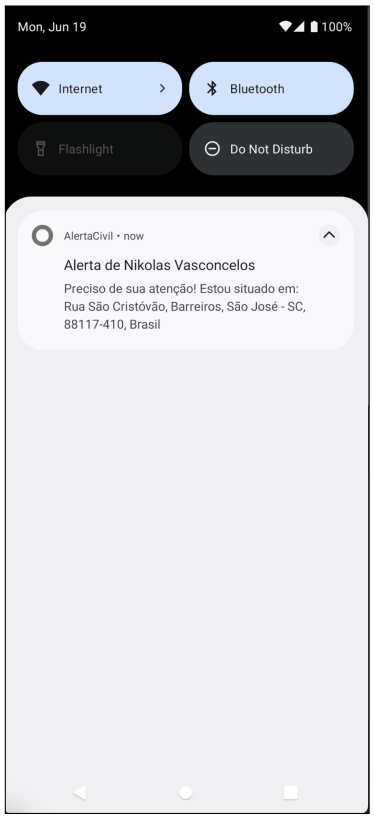


Figura 24 – Notificação recebida pelo contato com o app instalado

6 CONCLUSÃO

O objetivo principal deste trabalho foi desenvolver uma aplicação para dispositivos móveis, cujo principal fim é fornecer uma forma de pessoas se comunicarem de forma mais prática e rápida em momentos críticos onde cada segundo pode ser um diferencial. O desenvolvimento do aplicativo foi realizado utilizando a biblioteca React Native que, por ser uma tecnologia utilizada para desenvolvimento de aplicações híbridas, permitiu a entrega de uma aplicação para mais de uma plataforma sem a necessidade de desenvolver o mesmo aplicativo mais de uma vez.

A implementação das funcionalidades em React Native proporcionou uma solução eficiente e compatível com iOS e Android. No geral, o processo de desenvolvimento permitiu entregar um aplicativo móvel funcional, seguro e de fácil utilização, capaz de fornecer alertas de risco climático em tempo real e facilitar a comunicação entre os usuários e seus contatos selecionados.

Entretanto, devido a necessidade de as bibliotecas serem compatíveis com ambas as principais plataformas do mercado, como também dependem de empresas terceiras ou da comunidade para sua manutenção e publicação, em diversos momentos do desenvolvimento houve imprevisto onde a aplicação não mais era compilada e necessitou-se de manutenções durante a implementação.

Ainda assim conclui-se que, mesmo com tais situações controversas mencionados acima, o desenvolvimento da aplicação não foi impedido por estas, e certamente o tempo investido na implementação da solução foi consideravelmente inferior ao de implementar nativamente para cada plataforma, tendo assim obtido resultados satisfatórios.

7 TRABALHOS FUTUROS

Com base nos resultados obtidos após o desenvolvimento desta aplicação, bem como limitações encontradas e funcionalidades faltantes que foram identificadas no decorrer do desenvolvimento, os seguintes trabalhos futuros foram identificados após este trabalho:

- Novas funções na sessão de configurações, permitindo ao usuário:
  - Enviar ou não instantaneamente uma notificação a seus contatos favoritos ao ser identificado um alerta climático.
  - Definir uma mensagem padrão a ser enviada automaticamente ao pressionar o botão de alerta ao invés de redigi-la a cada vez que acessar a parte de SMS de seu telefone.
  - Implementar o envio de SMS automático (requer serviço pago).
- Uma aplicação web onde permite a um usuário administrador enviar mensagens específicas à população através deste aplicativo.

REFERÊNCIAS

ACTHERNOENE, Mathieu. **react-native-maps**. Github. Disponível em: https://www.npmjs.com/package/react-native-permissions. Acesso em: 3 jun. 2022.

AHMAD, Arshad *et al.* An empirical study of investigating mobile applications development challenges. **IEEE Access**, IEEE, v. 6, p. 17711–17728, 2018.

AIRBNB. **react-native-maps**. Airbnb. Disponível em: https://www.npmjs.com/package/react-native-maps. Acesso em: 3 jun. 2022.

AJAYI, Olusola Olajide *et al.* Performance evaluation of native and hybrid android applications. **Performance Evaluation**, v. 7, n. 16, 2018.

APPMASTER. **Como criar um aplicativo de alerta de emergência para uma vida segura usando No-Code?** AppMaster. 2022. Disponível em: https://appmaster.io/pt/blog/como-criar-um-aplicativo-de-alerta-de-emergencia. Acesso em: 4 jul. 2023.

CIVIL, Defesa. **Receber alertas da Defesa Civil por SMS**. Receber alertas da Defesa Civil por SMS. Disponível em: https://www.defesacivil.pr.gov.br/servicos/Seguranca/Defesa-Civil/Receber-alertas-da-Defesa-Civil-por-SMS-ERrZ4PN6. Acesso em: 12 mai. 2022.

EDSON KAIQUE LIMA, André Lucena. **Por que o escândalo mais recente do Facebook é mais grave que os anteriores?** Olhar Digital. 2021. Disponível em: https://olhardigital.com.br/2021/09/24/internet-e-redes-sociais/por-que-o-escandalo-mais-recente-do-facebook-e-mais-grave-que-os-anteriores/. Acesso em: 12 mai. 2022.

EXPO. **Make any app. Run it everywhere.** Expo. Disponível em: https://expo.dev. Acesso em: 20 jun. 2022.

FORONES, Jeffrey Marvin. **Authentication and Authorization**. Medium. 2021. Disponível em: https://medium.com/codex/native-vs-hybrid-mobile-app-development-cost-factors-a-breakdown-a97c196287c4. Acesso em: 4 mai. 2022.

GMBH, OpenCage. **Convert coordinates to and from places**. OpenCage GmbH. Disponível em: https://opencagedata.com. Acesso em: 10 jun. 2022.

GUNAWARDHANA, LK Pulasthi Dhananjaya. Native or Web or Hybridwhich is better for Mobile Application. **Turkish Journal of Computer and Mathematics Education (TURCOMAT)**, v. 12, n. 6, p. 4643–4649, 2021.

KUBBEN, Pieter. Mobile apps. **Fundamentals of Clinical Data Science**, Springer International Publishing, p. 171–179, 2019.

LIMITED, Invertase. **Cloud Messaging**. Invertase Limited. Disponível em: `https://rnfirebase.io/messaging/usage`. Acesso em: 3 jun. 2022.

_____. **Realtime Database**. Invertase Limited. Disponível em: `https://rnfirebase.io/database/usage`. Acesso em: 3 jun. 2022.

LLC, Google. **Sonhe, crie e transforme com o Google Cloud**. Google LLC. Disponível em: `https://cloud.google.com/?hl=pt-br`. Acesso em: 3 jun. 2022.

META PLATFORMS, Inc. **Como o Facebook sabe que eu estou em uma área afetada durante uma emergência?** Meta Platforms, Inc. Disponível em: `https://www.facebook.com/help/mobile-basic/778112215545209`. Acesso em: 12 mai. 2022.

METAOPENSOURCE. **Setting up the development environment**. Meta Platforms, Inc. Disponível em: `https://reactnative.dev/docs/environment-setup?guide=native&platform=ios`. Acesso em: 5 jun. 2022.

OPEN-METEO. **Weather Forecast API**. Open-Meteo. Disponível em: `https://open-meteo.com/en/docs`. Acesso em: 10 jun. 2022.

OVERFLOW, Stack. **2022 Developer Survey**. Stack Overflow. Disponível em: `https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment`. Acesso em: 3 jun. 2022.

RIFAT, Iftekhar. **react-native-geolocation-service**. Github. Disponível em: `https://www.npmjs.com/package/react-native-geolocation-service`. Acesso em: 3 jun. 2022.

SCHUCK, Sofia. **Relatório da ONU alerta para recordes das mudanças climáticas**. Redação Um Só Planeta. 2021. Disponível em: `https://umsoplaneta.globo.com/clima/noticia/2023/04/21/as-vesperas-do-dia-da-terra-relatorio-da-onu-alerta-para-recordes-das-mudancas-climaticas.ghtml`. Acesso em: 12 mai. 2022.

SCOTT, Chris. **react-native-background-fetch**. Transistor Software. Disponível em: `https://www.npmjs.com/package/react-native-background-fetch?activeTab=readme`. Acesso em: 3 jun. 2022.

SOUZA, Luiz Paulo. **Brasil tem mais smartphones do que habitantes, aponta levantamento**. Veja. 2023. Disponível em:

https://veja.abril.com.br/tecnologia/brasil-tem-mais-smartphones-do-que-habitantes-aponta-levantamento. Acesso em: 12 mai. 2022.

Desenvolvimento de software de localização para alertas de áreas de risco

Nikolas Damian Borges Vasconcelos

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) - Florianópolis, SC - Brazil

Abstract. *This work proposes a mobile app to notify people about severe weather events, such as storms, hurricanes, and tornadoes. The app aims to alert both locals and tourists to unusual climatic situations, reducing potential harm and losses. It will provide notifications based on the user's geographic location obtained from their smartphone. The app's effectiveness will be evaluated with the assistance of volunteer users.*

Resumo. *Este trabalho propõe um aplicativo móvel para notificar as pessoas sobre eventos climáticos severos, como tempestades, furacões e tornados. O aplicativo visa alertar moradores e turistas sobre situações climáticas incomuns, reduzindo possíveis danos e perdas. Ele fornecerá notificações com base na localização geográfica do usuário obtida em seu smartphone. A eficácia do aplicativo será avaliada com a ajuda de usuários voluntários.*

1. Introdução

Desastres climáticos como secas, inundações, ciclones e enchentes estão se tornando mais frequentes e intensos devido ao aumento do aquecimento global. A Organização das Nações Unidas alerta que essa tendência tende a se agravar com o passar do tempo. No Brasil, enchentes, alagamentos e inundações se tornaram parte do cotidiano durante as épocas chuvosas [Schuck 2021].

Com o crescente avanço da tecnologia, o uso de smartphones no Brasil está equiparado ao de países desenvolvidos. Estima-se que existam mais smartphones do que habitantes no país, com cerca de 249 milhões de aparelhos atualmente [Souza 2023]. Diante desse contexto, surge a necessidade de desenvolver uma solução para auxiliar na segurança da população.

Atualmente, a Defesa Civil de algumas regiões do estado de Santa Catarina envia mensagens de texto para pessoas cadastradas, informando sobre situações climáticas graves [Civil]. No entanto, muitas vezes, os parentes e entes queridos das pessoas próximas às regiões afetadas só ficam sabendo dessas situações no dia seguinte ou até mais tarde. Embora o Facebook possua uma funcionalidade semelhante, há preocupações quanto ao compartilhamento contínuo de dados de localização com a plataforma, devido aos escândalos envolvendo a empresa [Meta Platforms , Edson Kaique Lima 2021].

Existem soluções similares nos Estados Unidos, mas nenhum aplicativo no Brasil unifica todas as funcionalidades necessárias em um único sistema, especialmente adaptado às demandas e necessidades do país [AppMaster 2022]. Diante disso, propõe-se o desenvolvimento de um aplicativo móvel dedicado, que notifique os usuários sobre possíveis situações climáticas inesperadas e também permita a notificação imediata dos parentes e

entes queridos cadastrados, proporcionando uma resposta mais rápida e eficiente diante dos eventos climáticos.

Com base nessa necessidade, propõe-se o desenvolvimento de um aplicativo móvel que notifique os usuários sobre previsões climáticas adversas. Esse aplicativo seria direcionado tanto para residentes em áreas de risco quanto para turistas que estejam passando por essas regiões. Além disso, o aplicativo permitiria a notificação de familiares previamente cadastrados, para que todos possam tomar as medidas necessárias diante das situações climáticas.

2. Fundamentação teórica

Os avanços tecnológicos em dispositivos móveis têm transformado a interação das pessoas com a tecnologia. Com o aumento do uso de smartphones e tablets, é essencial desenvolver aplicativos móveis que atendam às necessidades dos usuários de maneira eficiente e intuitiva.

2.1. Aplicativos Móveis

De acordo com pesquisas, em 2018, iOS e Android juntos dominavam aproximadamente 99% do mercado de smartphones nos Estados Unidos, sendo iOS com 54% e Android com 45%. Existem abordagens diferentes para o desenvolvimento de aplicativos para cada plataforma, pois cada uma tem sua própria forma de desenvolvimento e publicação.

2.1.1. Aplicações Nativas

As aplicações nativas são desenvolvidas especificamente para uma plataforma específica, como Android ou iOS. Elas são escritas em linguagens de programação nativas, como Java ou Kotlin para Android e Objective-C ou Swift para iOS. Essas aplicações têm acesso total aos recursos do dispositivo, como câmera, GPS e sensores, permitindo a criação de experiências ricas e personalizadas.

As vantagens das aplicações nativas incluem desempenho otimizado, aproveitamento completo dos recursos do dispositivo e integração total com o sistema operacional. No entanto, o desenvolvimento de aplicações nativas requer conhecimento específico para cada plataforma, o que pode aumentar o tempo e o custo de desenvolvimento.

2.1.2. Aplicações Híbridas

Uma das principais vantagens das aplicações híbridas é a possibilidade de desenvolver uma única versão do aplicativo que funcione em várias plataformas, reduzindo o tempo e o custo de desenvolvimento. No entanto, as aplicações híbridas podem ter um desempenho ligeiramente inferior em comparação com as aplicações nativas e podem ter acesso limitado aos recursos do dispositivo, dependendo de implementações ou recursos de terceiros.

O desenvolvimento de aplicações híbridas utiliza tecnologias da web, como HTML, CSS e JavaScript, encapsuladas em um contêiner nativo. Esses aplicativos podem ser executados em diferentes plataformas, como Android e iOS, a partir de um único

código-base. Tecnologias populares para o desenvolvimento de aplicações híbridas incluem Apache Cordova, React Native e Flutter.

2.2. Acesso à Localização em Tempo Real

O acesso à localização em tempo real é uma funcionalidade importante em aplicativos móveis modernos, permitindo que os usuários obtenham informações personalizadas com base em sua localização. Com o GPS disponível na maioria dos dispositivos, o acesso à localização tornou-se uma funcionalidade básica em muitos aplicativos móveis.

O React Native oferece recursos nativos para acessar a localização em tempo real, facilitando sua implementação e uso.

2.3. Firebase

O Firebase é uma plataforma amplamente utilizada para o desenvolvimento de aplicativos móveis, oferecendo uma variedade de serviços para ajudar os desenvolvedores a criar, melhorar e expandir seus aplicativos. Alguns dos serviços oferecidos pelo Firebase são descritos a seguir.

2.3.1. Autenticação de Usuário

O Firebase Auth é um serviço de autenticação que permite aos desenvolvedores adicionar facilmente autenticação aos seus aplicativos móveis. Ele oferece métodos tradicionais de login, como email e senha, além de autenticação usando números de telefone.

A autenticação com número de telefone do Firebase Auth é uma solução segura e conveniente, eliminando a necessidade de lembrar senhas e oferecendo uma experiência de login rápida e simples.

2.3.2. Banco de Dados em Nuvem

O Firebase Firestore Database é um serviço de banco de dados NoSQL fornecido pelo Firebase. Ele permite que os desenvolvedores armazenem e sincronizem dados de forma eficiente em tempo real, facilitando a criação de aplicativos colaborativos e escaláveis.

O Firestore organiza os dados em coleções e documentos, permitindo consultas rápidas e eficientes. Além disso, oferece recursos avançados, sincronização em tempo real e integração com outros serviços do Firebase.

2.3.3. Notificações Push

O Firebase Cloud Messaging (FCM) é um serviço de mensagens em nuvem que permite aos desenvolvedores enviar notificações push para seus aplicativos móveis. As notificações push são entregues diretamente aos dispositivos dos usuários, mesmo quando o aplicativo não está em execução.

O FCM oferece uma solução poderosa para o envio de notificações push em tempo real, permitindo segmentação de usuários e envio de mensagens personalizadas.

3. Projeto de Solução

Nesta seção, apresentamos uma proposta de aplicativo móvel para iOS e Android utilizando a tecnologia React Native. O objetivo do aplicativo é fornecer alertas de risco climático em tempo real com base na localização do usuário, além de permitir o envio de mensagens de texto (SMS) e notificações push para contatos selecionados. O banco de dados será implementado usando o Realtime Database do Firebase, e o login será realizado através do número de telefone do usuário utilizando o Firebase Auth.

O aplicativo proposto oferecerá uma solução prática e eficiente para receber alertas de risco climático em tempo real. A implementação com React Native e Firebase permitirá um desenvolvimento rápido e a integração com serviços robustos de geolocalização, autenticação, armazenamento de dados e envio de notificações.

3.1. Caso de Uso

Foi elaborado um diagrama de caso de uso para analisar as necessidades e requisitos da aplicação. O principal ator é o usuário do aplicativo, que dispara os alertas de incidente climático. O diagrama de caso de uso mostra as interações do usuário com o aplicativo e identifica três atores externos. A Figura 1 apresenta o diagrama de caso de uso com o principal ator sendo o usuário do aplicativo.

3.2. Diagrama de Fluxo

Foi elaborado um diagrama de fluxo que representa o fluxo completo da aplicação, desde a abertura até o envio da mensagem aos contatos favoritados. O diagrama de fluxo apresentado na Figura 2 mostra todas as etapas do fluxo, incluindo a autenticação do usuário, a tela principal do mapa, a seleção de contatos favoritados e o envio de mensagens ou notificações.

3.3. Arquitetura da Aplicação

Com base nos diagramas anteriores e nas necessidades do projeto, foi elaborado um diagrama de arquitetura da aplicação. A Figura 3 mostra a arquitetura da aplicação, onde o elemento central é o aplicativo do usuário, conectado a serviços externos, como os serviços do Google Cloud e serviços implementados que utilizam a localização do usuário. Esses serviços se comunicam com o aplicativo e entre si, conforme representado no diagrama.

3.4. Mapa com Acesso à Localização em Tempo Real

O aplicativo utilizará a API de localização do dispositivo para rastrear a posição do usuário em tempo real. Será exibido um mapa interativo utilizando a biblioteca react-native-maps, mostrando a localização atual do usuário e alertas de risco climático próximos.

3.5. Envio de Mensagens

O aplicativo permitirá que o usuário envie mensagens para contatos selecionados. O usuário poderá escolher contatos de sua lista de contatos do dispositivo e enviar alertas de risco ou informações importantes através de SMS. O envio de SMS será realizado usando a feature de Linking do React Native, permitindo que o usuário acesse o sistema de envio de SMS do dispositivo.

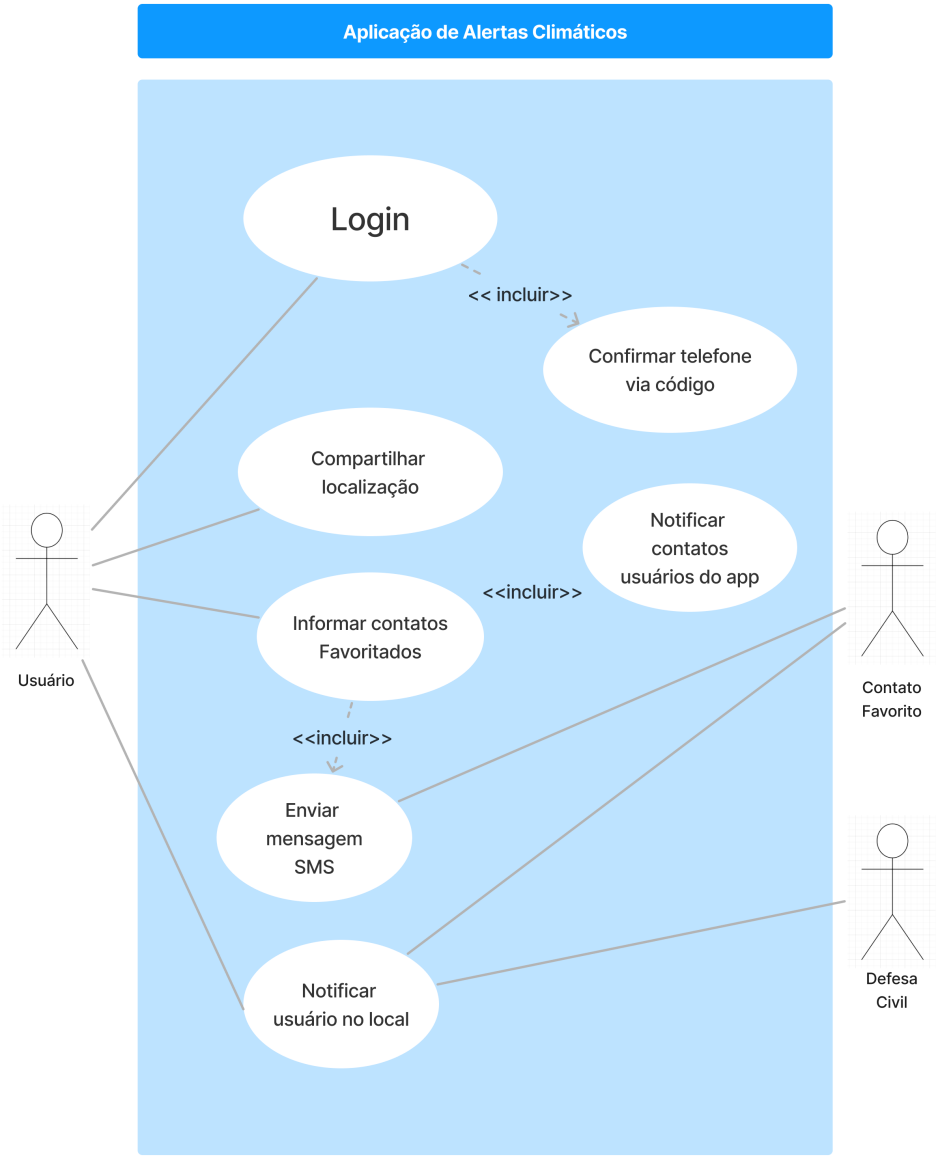


Figure 1. Diagrama de Casos de Uso

3.6. Banco de Dados na Nuvem e Local

O Firebase será utilizado como plataforma de banco de dados para armazenar informações relevantes para o funcionamento do aplicativo. O Realtime Database do Firebase será usado para armazenar informações localmente no formato JSON e sincronizá-las com o banco na nuvem. O Firebase também oferece recursos de segurança e backup para o banco de dados.

3.7. Login com Número de Telefone usando o Firebase

O Firebase Authentication será utilizado para autenticação e login dos usuários. A opção de login através do número de telefone será implementada, permitindo que os usuários acessem o aplicativo utilizando seu número de telefone, aumentando a segurança e facilitando o processo de login.

3.8. Alertas de Risco Climático Baseado na Localização do Usuário

Utilizando a API Open Meteo, o aplicativo será capaz de alertar os usuários sobre possíveis riscos em sua localização atual. Alertas e notificações serão exibidos para que o usuário esteja ciente de condições meteorológicas perigosas e possa tomar as medidas apropriadas para sua segurança.

3.9. Notificações Push

Será implementada a funcionalidade de envio de notificações push para os usuários do aplicativo. Utilizando o Firebase Cloud Messaging (FCM), será possível enviar notificações instantâneas para alertar os usuários sobre riscos climáticos iminentes ou informações relevantes.

3.10. Contatos

A lista de contatos do usuário será a mesma lista do dispositivo. O aplicativo terá acesso de leitura aos contatos, mas não armazenará nem compartilhará essas informações. O usuário poderá selecionar contatos favoritados para enviar mensagens ou notificações, e o aplicativo buscará se esses contatos também são usuários do aplicativo.

3.11. Localização

O aplicativo não armazenará a localização do usuário em nenhum momento. A localização será usada apenas para mostrar ao usuário sua posição no mapa e compartilhá-la com contatos selecionados, se desejado. O compartilhamento da localização será feito utilizando a API da OpenCage para obter a descrição do local e enviar uma mensagem compreensível ao usuário.

4. Desenvolvimento

Nesta sessão, será apresentada a metodologia adotada para o desenvolvimento e implantação do aplicativo para dispositivos móveis utilizando a biblioteca React Native. Serão descritos os procedimentos e etapas realizadas ao longo do processo, a implementação do aplicativo e a sua implantação. Também serão abordadas as tecnologias utilizadas.

4.1. Preparação do Ambiente de desenvolvimento

Para o desenvolvimento de aplicações utilizando React Native, é necessário preparar o ambiente de desenvolvimento. A utilização da ferramenta Expo é recomendada para desenvolvedores com pouco contato com a tecnologia, pois permite a construção rápida de aplicativos, exigindo apenas o Node.js e um telefone ou emulador da plataforma desejada.

4.1.1. React Native CLI vs. Expo

Neste projeto, foi utilizado o React Native CLI, que oferece mais flexibilidade e compatibilidade com as versões mais recentes do React Native. Ele permite a configuração nativa completa e produz um aplicativo final mais leve. Além disso, algumas APIs nativas, como o Bluetooth, não funcionam com o Expo. O editor de texto Visual Studio Code foi escolhido para o desenvolvimento principal da aplicação.

4.2. Estrutura do Projeto

A estrutura do projeto React Native segue uma organização padrão, com pastas específicas para cada plataforma (iOS e Android) e uma pasta ”src” para o desenvolvimento principal da aplicação.

4.3. Permissões

A aplicação requer acesso a recursos nativos do dispositivo, como localização em tempo real, lista de contatos, envio de SMS e recebimento de notificações. Para isso, é utilizada a biblioteca react-native-permissions, que permite solicitar ao usuário as permissões necessárias para o correto funcionamento da aplicação.

4.4. Interface da Aplicação

A aplicação possui diversas telas desenvolvidas, incluindo tela de login, tela de confirmação, tela principal com mapa, tela de configurações e tela de contatos. A interação com o mapa e a exibição de contatos favoritos são implementadas utilizando o componente Bottom Sheet. As telas são desenvolvidas utilizando os recursos do React Native.

4.5. Integrações

4.5.1. Firebase

O Firebase é utilizado para o recebimento de notificações e o armazenamento de dados dos usuários. A biblioteca react-native-firebase/messaging é utilizada para a comunicação com os serviços nativos de notificação do Firebase. Já a biblioteca react-native-firebase/database é utilizada para armazenar os números de telefone dos usuários e seus tokens de notificação no Realtime Database.

4.5.2. Alertas Climáticos

A aplicação utiliza a API gratuita e Open Source da plataforma Open Meteo para implementar os alertas climáticos. Através dessa API, é possível obter informações climáticas com base na localização do usuário.

4.5.3. Mensagens via SMS

A aplicação permite o envio de mensagens via SMS para contatos favoritos. Essa funcionalidade é implementada utilizando a biblioteca Linking do React Native, que permite acessar a função nativa de envio de SMS do dispositivo.

4.5.4. Localização

Através da API de geolocalização da OpenCage, é possível obter informações detalhadas sobre a localização do usuário. Essa integração é realizada através de requisições REST para obter descrições da localização com base em latitude e longitude.

4.6. Implantação

A aplicação não foi implantada, pois o objetivo deste trabalho é disponibilizá-la para instituições interessadas em reutilizá-la. Caso haja interesse em implantar a aplicação, é possível fazê-lo por meio das plataformas Google Play Console (para dispositivos Android) e App Store Connect (para dispositivos iOS).

5. Avaliação Funcional

Após o desenvolvimento da aplicação e suas funcionalidades, foram realizados testes para confirmar sua efetividade. Os testes foram feitos em um simulador de dispositivo Android, em um notebook Macbook Pro 2023, e em um smartphone com Android 13.

5.1. Login

A funcionalidade de login, implementada com integração ao Firebase Auth, foi testada e mostrou-se simples e fluida. Foram testados os campos de telefone e código de verificação, como mostrado nas Figuras 4 e 5.

5.2. Alertas enviados aos contatos

Os alertas climáticos foram testados utilizando as APIs Open Meteo e Open Cage. Foram simuladas situações de alerta e os usuários marcados como favoritos foram notificados através do aplicativo. Foram realizados testes de notificação em segundo plano, com o aplicativo fechado (Figura 6), e com o aplicativo aberto (Figura 7). Também foi possível exibir a localização do incidente no mapa do aplicativo (Figura 8).

5.3. Alertas via SMS

Os alertas também foram enviados via SMS para os contatos favoritos. Foi possível abrir o aplicativo de SMS padrão do dispositivo, preencher a mensagem e enviar para os contatos marcados como favoritos. Os detalhes da localização foram incluídos na mensagem. As Figuras 9, 10 e 11 mostram o processo de envio das mensagens via SMS. O contato com o aplicativo instalado também recebeu uma notificação push juntamente com a mensagem SMS (Figura 12).

A avaliação funcional da aplicação mostrou que as funcionalidades foram implementadas corretamente e estão operando conforme o esperado.

6. Conclusão

O presente artigo apresenta a conclusão de um trabalho cujo objetivo principal foi desenvolver um aplicativo móvel para permitir uma comunicação mais prática e rápida em momentos críticos. A aplicação foi desenvolvida utilizando a biblioteca React Native, o que possibilitou a entrega do aplicativo em múltiplas plataformas sem a necessidade de desenvolvê-lo novamente.

A implementação das funcionalidades em React Native foi eficiente e compatível com iOS e Android. O processo de desenvolvimento resultou em um aplicativo móvel funcional, seguro e fácil de usar, que oferece alertas de risco climático em tempo real e facilita a comunicação entre os usuários e seus contatos selecionados.

No entanto, durante o desenvolvimento, surgiram imprevistos devido à necessidade de compatibilidade com diferentes plataformas e dependência de bibliotecas mantidas por terceiros ou pela comunidade. Isso exigiu manutenções adicionais durante a implementação.

Apesar dessas situações controversas mencionadas, o desenvolvimento da aplicação não foi impedido, e o tempo investido na implementação da solução foi consideravelmente menor em comparação com o desenvolvimento nativo para cada plataforma. Como resultado, os objetivos do trabalho foram alcançados de forma satisfatória.

7. Trabalhos futuros

Com base nos resultados obtidos após o desenvolvimento desta aplicação, bem como limitações encontradas e funcionalidades faltantes que foram identificadas no decorrer do desenvolvimento, os seguintes trabalhos futuros foram identificados após este trabalho:

- Novas funções na sessão de configurações, permitindo ao usuário:
  - Enviar ou não instantaneamente uma notificação a seus contatos favoritos ao ser identificado um alerta climático.
  - Definir uma mensagem padrão a ser enviada automaticamente ao pressionar o botão de alerta ao invés de redigi-la a cada vez que acessar a parte de SMS de seu telefone.
  - Implementar o envio de SMS automático (requer serviço pago).
- Uma aplicação web onde permite a um usuário administrador enviar mensagens específicas à população através deste aplicativo.

References

AppMaster (2022). Como criar um aplicativo de alerta de emergência para uma vida segura usando no-code?

Civil, D. Receber alertas da defesa civil por sms.

Edson Kaique Lima, A. L. (2021). Por que o escândalo mais recente do facebook é mais grave que os anteriores?

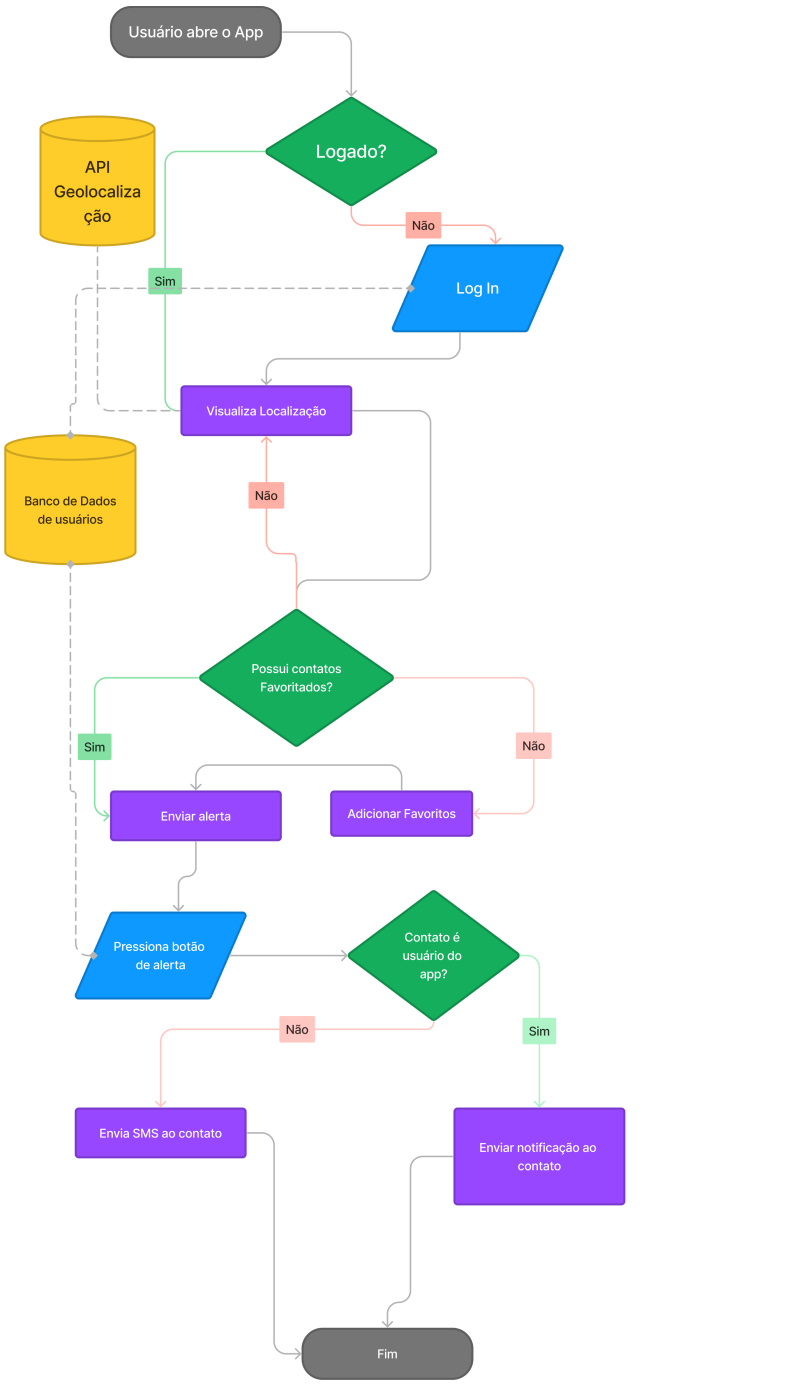
Meta Platforms, I. Como o facebook sabe que eu estou em uma área afetada durante uma emergência?

Schuck, S. (2021). Relatório da onu alerta para recordes das mudanças climáticas.

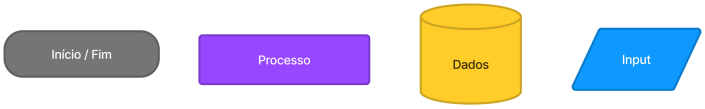
Souza, L. P. (2023). Brasil tem mais smartphones do que habitantes, aponta levantamento.

Estória de Usuário

Como usuário, gostaria de notificar pessoas escolhidas previamente quando acontecer algo ao meu redor referente a instabilidades climáticas ou questões de saúde, como também informá-los onde estou



Legenda



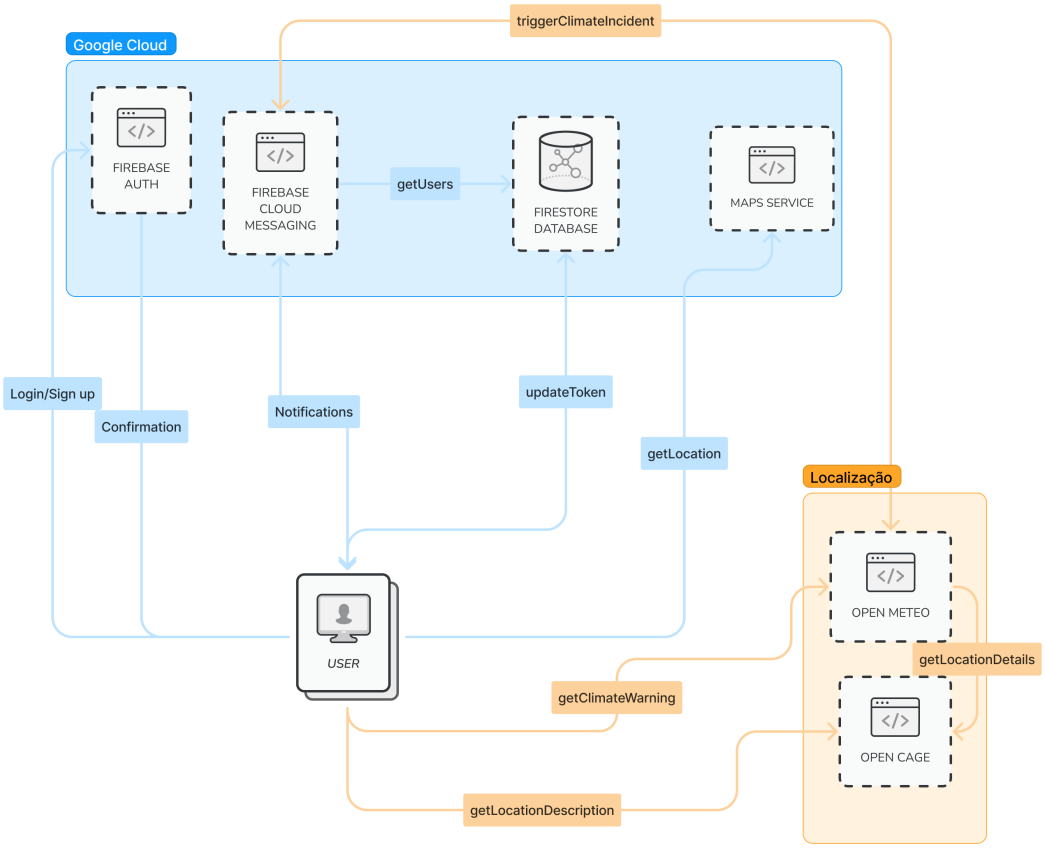


Figure 3. Diagrama de Arquitetura

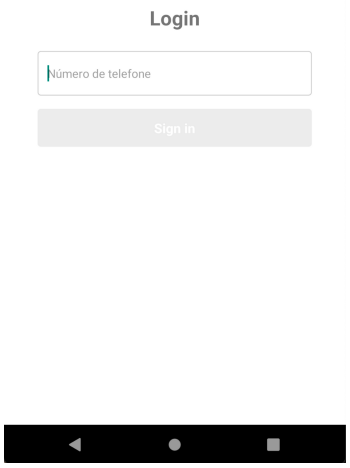


Figure 4. Tela Login com campo vazio

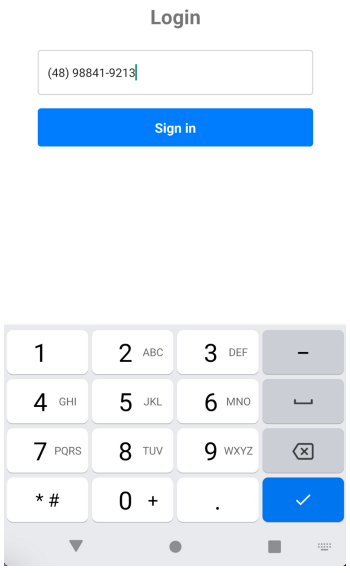


Figure 5. Tela de login com campo preenchido

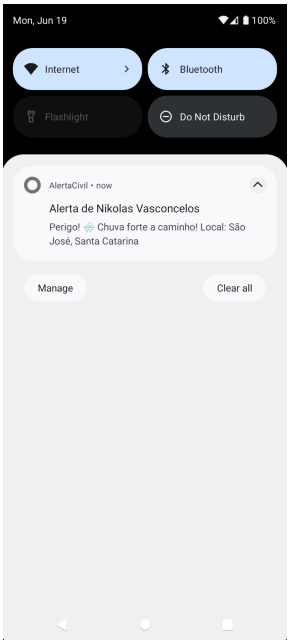


Figure 6. Notificação em segundo plano

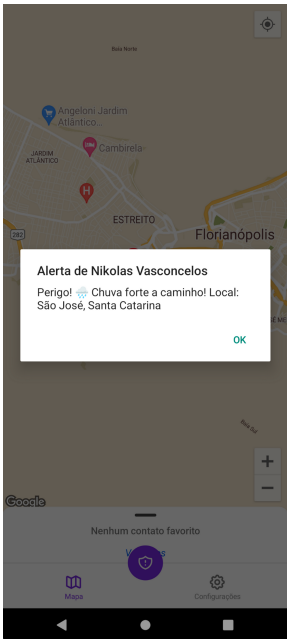


Figure 7. Notificação com aplicativo aberto

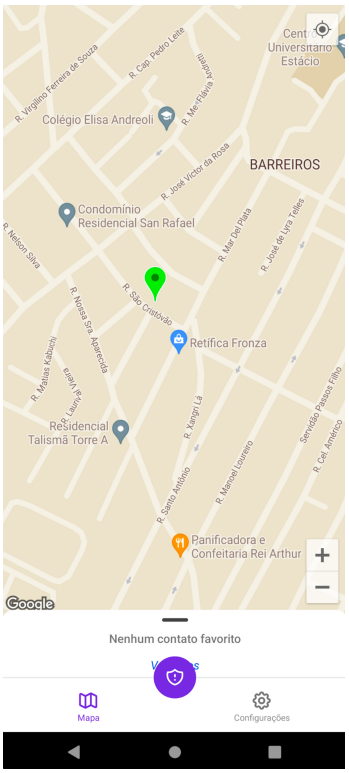


Figure 8. Localização advinda da notificação recebida

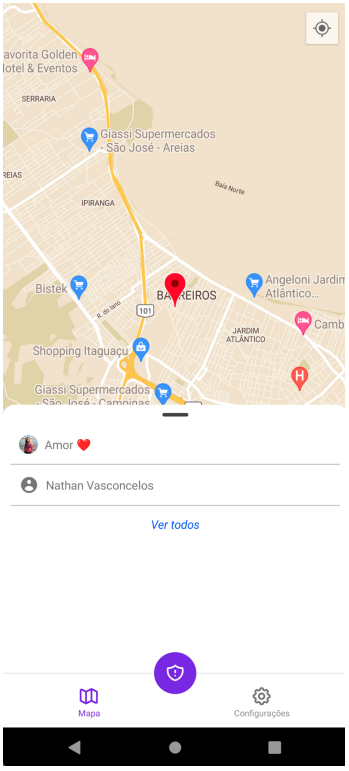


Figure 9. Contatos marcados como favoritos

Figure 10. Aplicativo de SMS aberto com a mensagem no input de texto

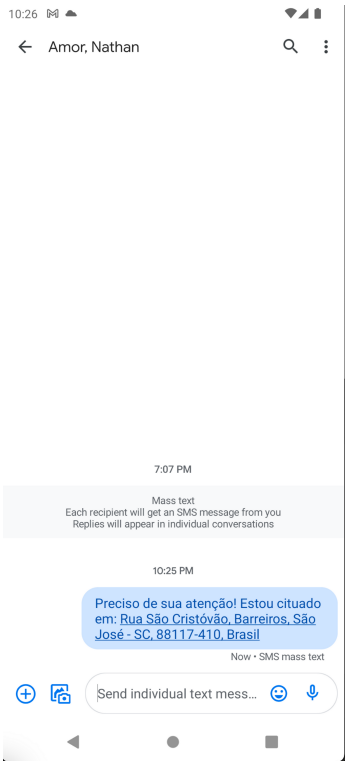


Figure 11. Aplicativo de SMS aberto com a mensagem enviada aos contatos

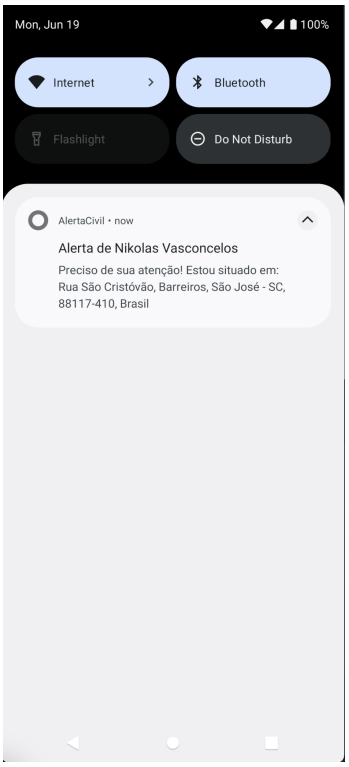


Figure 12. Notificação recebida pelo contato com o app instalado

```
// tsconfig.json

// prettier-ignore
{
  "extends": "@tsconfig/react-native/tsconfig.json", /* Recommended
     React Native TSConfig base */
    "compilerOptions": {
    /* Visit https://aka.ms/tsconfig.json to read more about this file
       */
    /* Completeness */
    "skipLibCheck": true, /* Skip type checking all .d.ts files. */
  }
}

// package.json

{
  "name": "AlertaCivil",
    "version": "0.0.1",
      "private": true,
        "scripts": {
    "android": "react-native run-android",
      "android:clean": "cd android;./gradlew clean;cd ..",
        "ios": "react-native run-ios",
          "start": "react-native start --reset-cache",
            "test": "jest",
              "lint": "eslint . --ext .js,.jsx,.ts,.tsx"
  },
  "dependencies": {
    "@brazilian-utils/brazilian-utils": "^1.0.0-rc.12",
      "@gorhom/bottom-sheet": "^4.4.3",
        "@react-native-async-storage/async-storage": "^1.18.2",
          "@react-native-community/geolocation": "^3.0.6",
            "@react-native-firebase/app": "16.4.0",
              "@react-native-firebase/auth": "16.4.0",
                "@react-native-firebase/database": "^18.0.0",
                  "@react-native-firebase/firestore": "16.4.0",
                    "@react-native-firebase/messaging": "16.4.0",
                      "@react-native-google-signin/google-signin": "
                         ^8.1.0",
                        "@react-navigation/bottom-tabs": "^6.3.1",
                          "@react-navigation/native": "^6.0.10",
                            "@react-navigation/stack": "^6.2.1",
                              "@shopify/flash-list": "^1.2.2",
                                "axios": "^1.4.0",
                                  "expo": "^47.0.0",
                                    "expo-modules-core": "^1.2.7",
```

```
45                                    "expo-sms": "^11.2.1",
46                                      "firebase-admin": "^11.2.0",
47                                        "firebase-functions": "^4.0.2"
                                             ,
48                                          "react": "18.1.0",
49                                            "react-fast-compare": "
                                                 ^3.2.0",
50                                              "react-native": "0.70.5"
                                                   ,
51                                                "react-native-
                                                     confirmation-code-
                                                     field": "^7.3.1",
52                                                 "react-native-
                                                      contacts": "
                                                      ^7.0.5",
53                                                  "react-native-
                                                       device-info": "
                                                       ^10.0.2",
54                                                   "react-native-
                                                        geolocation-
                                                        service": "
                                                        ^5.3.0",
55                                                    "react-native-
                                                         gesture-
                                                         handler": "
                                                         ^2.5.0",
56                                                     "react-
                                                          native-
                                                          heroicons
                                                          ": "
                                                          ^3.2.0",
57                                                      "react-
                                                           native-
                                                           maps":
                                                           "^1.3.1
                                                           ",
58                                                       "react-
                                                            native
                                                            -
                                                            permissions
                                                            ": "
                                                            ^3.6.1
                                                            ",
59                                                        "react
                                                             -
                                                             native
                                                             -
```

```
phone-input": "^1.3.4",
"react-native-reanimated": "^2.10.0",
"react-native-safe-area-context": "^4.5.3",
"react-native-screens":
```

```
63
64
65
66  },
67  "devDependencies": {
68    "@babel/core": "^7.12.9",
```

```
69      "@babel/plugin-proposal-nullish-coalescing-operator": "^7.18.6",
70        "@babel/plugin-proposal-optional-chaining": "^7.18.9",
71          "@babel/plugin-transform-arrow-functions": "^7.18.6",
72            "@babel/plugin-transform-shorthand-properties": "^7.18.6",
73              "@babel/plugin-transform-template-literals": "^7.18.9",
74                "@babel/runtime": "^7.12.5",
75                  "@react-native-community/eslint-config": "^2.0.0",
76                    "@tsconfig/react-native": "^2.0.2",
77                      "@types/jest": "^26.0.23",
78                        "@types/lodash": "^4.14.195",
79                          "@types/react": "^18.0.21",
80                            "@types/react-native": "^0.70.6",
81                              "@types/react-native-dotenv": "^0.2.0",
82                                "@types/react-test-renderer": "^18.0.0",
83                                  "@typescript-eslint/eslint-plugin": "
                                       ^5.37.0",
84                                    "@typescript-eslint/parser": "
                                         ^5.37.0",
85                                     "babel-jest": "^26.6.3",
86                                       "eslint": "^7.32.0",
87                                         "jest": "^26.6.3",
88                                           "metro-react-native-babel-
                                              preset": "0.72.3",
89                                          "react-native-dotenv": "
                                               ^3.4.8",
90                                           "react-test-renderer": "
                                                18.1.0",
91                                            "typescript": "^4.8.3"
92    },
93    "jest": {
94      "preset": "react-native",
95        "moduleFileExtensions": [
96          "ts",
97          "tsx",
98          "js",
99          "jsx",
100         "json",
101         "node"
102       ]
103   }
104 }
105
106
107 \textbf{ metro.config.json }
108 /**
109 * Metro configuration for React Native
110 * https://github.com/facebook/react-native
```

```
*
* @format
*/

module.exports = {
  transformer: {
    getTransformOptions: async () => ({
      transform: {
        experimentalImportSupport: false,
        inlineRequires: true,
      },
    }),
  },
};


\textbf{ index.js }

/**
 * @format
 */

import { AppRegistry } from 'react-native';
import App from './src/App';
import { name as appName } from './app.json';

AppRegistry.registerComponent(appName, () => App);


\textbf{ firebase.json }

{
  "functions": [
    {
      "source": "functions",
      "codebase": "default"
    },
    {
      "source": "alerta-civil-firebase",
      "codebase": "alerta-civil-firebase",
      "ignore": [
        "node_modules",
        ".git",
        "firebase-debug.log",
        "firebase-debug.*.log"
      ],
      "predeploy": [
```

```
        "npm --prefix \"$RESOURCE_DIR\" run lint",
        "npm --prefix \"$RESOURCE_DIR\" run build"
      ]
    }
  ],
    "database": {
    "rules": "database.rules.json"
  }
}

\textbf{ database.rules.json }

{
  "rules": {
    ".read": "now < 1688439600000",
      ".write": "now < 1688439600000"
  }
}

\textbf{ babel.config.js }

module.exports = {
  presets: ['module:metro-react-native-babel-preset'],
  plugins: [
    'react-native-reanimated/plugin',
    '@babel/plugin-transform-shorthand-properties',
    '@babel/plugin-proposal-optional-chaining',
    '@babel/plugin-proposal-nullish-coalescing-operator',
    '@babel/plugin-transform-template-literals'
  ],
};


\textbf{ app.json }

{
  "name": "AlertaCivil",
    "displayName": "AlertaCivil"
}

\textbf{.ruby - version }

2.7.5

\textbf{.prettierrs.js }

module.exports = {
```

```
  arrowParens: 'avoid',
  bracketSameLine: true,
  bracketSpacing: true,
  singleQuote: true,
  trailingComma: 'all',
};



\textbf{.gitignore }


    # OSX
#
  .DS_Store

# Xcode
#
build /
*.pbxuser
!default.pbxuser
  *.mode1v3
!default.mode1v3
  *.mode2v3
!default.mode2v3
  *.perspectivev3
!default.perspectivev3
xcuserdata
  *.xccheckout
  *.moved - aside
DerivedData
  *.hmap
  *.ipa
  *.xcuserstate
ios /.xcode.env.local

# Android / IntelliJ
#
build /
.idea
    .gradle
local.properties
  *.iml
  *.hprof

# node.js
#
```

```
252 node_modules /
253   npm - debug.log
254 yarn - error.log
255
256 # BUCK
257 buck - out /
258 \.buckd /
259 *.keystore
260 !debug.keystore
261
262 # fastlane
263 #
264 # It is recommended to not store the screenshots in the git repo.Instead
      , use fastlane to re - generate the
265 # screenshots whenever they are needed.
266 # For more information about the recommended setup visit:
267 # https://docs.fastlane.tools/best-practices/source-control/
268
269 ** /fastlane/report.xml
270   ** /fastlane/Preview.html
271   ** /fastlane/screenshots
272   ** /fastlane/test_output
273
274 # Bundle artifact
275   *.jsbundle
276
277 # Ruby / CocoaPods
278   / ios / Pods /
279   /vendor/bundle /
280
281 # iCloud
282
283   *.icloud
284
285
286     .firebaserc
287
288
289 {
290   "projects": {
291     "default": "alertacivil-e00c3"
292   }
293 }
294
295
296
297 .eslintrc
```

```
298
299
300 {
301   "root": true,
302     "extends": [
303       "@react-native-community"
304     ],
305       "parser": "@typescript-eslint/parser",
306         "plugins": [
307           "@typescript-eslint",
308           "import"
309         ],
310           "env": {
311     "jest": true,
312       "jasmine": true
313   },
314   "globals": {
315     "debugOutput": true,
316       "__EMULATOR__": true,
317         "__DEV__": true,
318           "__PROD__": true,
319             "__STAGING__": true,
320               "__E2E__": true,
321                 "__CLI__": true,
322                   "__JEST__": true,
323                     "BUNDLE_ID": true,
324                       "VERSION": true,
325                         "VERSION_NAME_FULL": true,
326                           "VERSION_NAME_SHORT": true,
327                             "BUILD_VERSION": true,
328                               "CODE_VERSION": true,
329                                 "ENV_STR": true,
330                                   "VERSION_STR": true,
331                                     "__REDUX_DEVTOOLS_EXTENSION__": true
                                        ,
332                                       "_global": true,
333                                         "fetch": true,
334                                           "window": true
335   },
336   "rules": {
337     "react-native/no-unused-styles": 2
338     // "react-native/split-platform-components": 2,
339     // "react-native/no-inline-styles": 2,
340     // "react-native/no-color-literals": 2,
341     // "react-native/no-raw-text": 2,
342     // "react-native/no-single-element-style-arrays": 2
343   },
```

```
344   "overrides": [
345     {
346       "files": [
347         "app/**/*.test?(s).(j|t)s(x?)"
348       ],
349       "env": {
350         "jest": true
351       },
352       "rules": {
353         "global-require": 0,
354         "import/first": 0
355       }
356     },
357     {
358       "files": [
359         "e2e/**/*.js"
360       ],
361       "extends": [
362         "@react-native-community",
363         "plugin:wdio/recommended"
364       ],
365       "plugins": [
366         "@typescript-eslint",
367         "import",
368         "wdio"
369       ],
370       "env": {
371         "jasmine": true
372       },
373       "globals": {
374         "element": true,
375         "by": true,
376         "waitFor": true,
377         "device": true,
378         "TEST_CONFIGS": true
379       }
380     },
381     {
382       "files": [
383         "e2e_appium/**/*.ts"
384       ],
385       "globals": {
386         "$": true
387       }
388     },
389     {
390       "files": [
```

```
        "*.tsx"
      ],
      "rules": {
        "semi": 0,
        "import/order": [
          "error",
          {
            "groups": [
              [
                "builtin",
                "external"
              ],
              [
                "internal",
                "parent"
              ],
              "sibling",
              "index"
            ]
          }
        ]
      }
    }
    // {
    //   "files": ["*.js"],
    //   "rules": { "@typescript-eslint/explicit-function-return-type":
        0 }
    // }
  ]
}


.env


OPENCAGE_KEY = 925636d1563141828ff46bd081f070dc


  .buckconfig



[android]
target = Google Inc.:Google APIs: 23

[maven_repositories]
central = https://repo1.maven.org/maven2
```

```
437
438
439
440 .babelrc
441
442
443 {
444   "presets": [
445     "module:metro-react-native-babel-preset",
446     "module:react-native-dotenv"
447   ],
448     "plugins": [
449       "module:react-native-dotenv"
450     ]
451 }
452
453
454 /src/components / CodeVerification / CodeVerification.tsx
455
456
457 import React, { useEffect, useState } from 'react';
458 import {
459   View,
460   Text,
461   TouchableOpacity,
462   StyleSheet,
463 } from 'react-native';
464 import {
465   CodeField,
466   Cursor,
467   useBlurOnFulfill,
468   useClearByFocusCell,
469 } from 'react-native-confirmation-code-field';
470 import { ScreenContainer } from '../Shared/ScreenContainer'
471 import { ScreenTitle } from '../Shared/ScreenTitle';
472 import useAuth from '../../hooks/useAuth';
473
474 const CELL_COUNT = 6;
475
476 const CodeVerification = ({ navigation }: any) => {
477   const { confirmCode } = useAuth();
478   const [code, setCode] = useState('');
479   const ref = useBlurOnFulfill({ value: code, cellCount: CELL_COUNT });
480   const [props, getCellOnLayoutHandler] = useClearByFocusCell({
481     value: code,
482     setValue: setCode,
483   });
```

```
484
485   const handleVerifyCode = async () => {
486     confirmCode(code);
487   };
488
489   useEffect(() => {
490     if (code.length === CELL_COUNT) {
491       handleVerifyCode()
492     }
493   }, [code])
494
495   return (
496     <ScreenContainer>
497     <ScreenTitle>Verifica  o < /ScreenTitle>
498     < Text style = { StyleSheet.flatten([styles.subtitle, styles.padding
          ]) } >
499       Enviamos um SMS para o seu n mero.Insira  abaixo o c digo
          recebido.
500       < /Text>
501         < CodeField
502   ref = { ref }
503   {...props }
504   // Use `caretHidden={false}` when users can't paste a text value,
       because context menu doesn't appear
505   value = { code }
506   onChangeText = { setCode }
507   cellCount = { CELL_COUNT }
508   rootStyle = { StyleSheet.flatten([styles.codeFieldRoot, styles.padding
       ]) }
509   keyboardType = "number-pad"
510   textContentType = "oneTimeCode"
511   renderCell = {({ index, symbol, isFocused }) => (
512     <Text
513             key= { index }
514 style = { [styles.cell, isFocused && styles.focusCell]}
515 onLayout = { getCellOnLayoutHandler(index) } >
516   { symbol || (isFocused ? <Cursor /> : null)}
517     < /Text>
518       )}
519 />
520   < TouchableOpacity style = { StyleSheet.flatten([styles.button, styles
       .padding]) } onPress = { handleVerifyCode } >
521     <Text style={ styles.buttonText }> Continuar < /Text>
522       < /TouchableOpacity>
523       < /ScreenContainer>
524   );
525 };
```

```
526
527 export default CodeVerification;
528
529 const styles = StyleSheet.create({
530   padding: {
531     paddingHorizontal: 12,
532   },
533   title: {
534     fontSize: 24,
535     fontWeight: 'bold',
536     marginBottom: 16,
537   },
538   subtitle: {
539     fontSize: 16,
540     marginTop: 32,
541   },
542   input: {
543     backgroundColor: '#F2F2F2',
544     borderRadius: 8,
545     padding: 16,
546     marginBottom: 16,
547     fontSize: 16,
548   },
549   button: {
550     backgroundColor: '#007AFF',
551     borderRadius: 8,
552     padding: 16,
553     alignItems: 'center',
554   },
555   buttonText: {
556     color: '#FFFFFF',
557     fontSize: 16,
558     fontWeight: 'bold',
559   },
560   codeFieldRoot: { marginBottom: 20, flex: 1, alignItems: 'center' },
561   cell: {
562     width: 40,
563     height: 40,
564     lineHeight: 38,
565     fontSize: 24,
566     borderWidth: 2,
567     borderColor: '#00000030',
568     textAlign: 'center',
569   },
570   focusCell: {
571     borderColor: '#000',
572   },
```

```
});


\textbf{ /src/components / CodeVerification / index.ts }


export { default as CodeVerification } from './CodeVerification'


\textbf{ /src/components / ContactsScreen / ContactsScreen.tsx }


import React, { memo, useState } from 'react'
import { View, StyleSheet, TextInput } from 'react-native'
import isEqual from 'react-fast-compare';
import { ScreenContainer } from '../Shared/ScreenContainer';
import { ScreenTitle } from '../Shared/ScreenTitle';
import { ContactsList } from '../Shared/ContactsList';
import { commonColors } from '../../theme/styles';
import { useContacts } from '../../hooks/useContacts';


const ContactsScreen = () => {
  const { contacts } = useContacts();
  const [contactSearch, setContactSearch] = useState('')

  return (
    <ScreenContainer style= { styles.container } >
    <ScreenTitle>Contatos < /ScreenTitle>
    < View style = { styles.listWrapper } >
      <TextInput placeholder='Buscar' textAlignVertical = 'top' style =
         { styles.input } onChangeText = { setContactSearch } />
        <ContactsList filter={ contactSearch } data = { contacts } />
          </View>
          < /ScreenContainer>
  )
}

export default memo(ContactsScreen, isEqual)

const styles = StyleSheet.create({
  container: { paddingBottom: 0 },
  listWrapper: {
    flex: 1,
    paddingTop: 28
  },
  input: {
```

```
    height: 40,
    borderWidth: 1,
    borderColor: commonColors.color5,
    borderRadius: 10,
    paddingHorizontal: 10,
    marginBottom: 25,
    marginHorizontal: 10,
  }
})


\textbf{ /src/components / ContactsScreen / index.ts }

export { default as ContactsScreen } from './ContactsScreen'

\textbf{ /src/components / FavoritesSheet / FavoritesSheet.tsx }


import React, { memo, useCallback, useMemo, useRef } from 'react';
import { View, StyleSheet } from 'react-native';
import BottomSheet from '@gorhom/bottom-sheet';
import { commonColors } from '../../theme/styles';
import { ContactsList } from '../Shared/ContactsList';
import isEqual from 'react-fast-compare';
import { useContacts } from '../../hooks/useContacts';

const snaps = ['15%', '20%', '40%']

const FavoritesSheet = () => {
  const bottomSheetRef = useRef<BottomSheet>(null);
  const { favorites: favoritesList } = useContacts();
  const DATA = useMemo(
    () => [
      ...favoritesList,
    ],
    [favoritesList],
  );
  const hasData = DATA.length > 0
  const snapPoints = useMemo(() => {
    if (hasData) {
      return snaps;
    }
    snaps.unshift('10%')
    return snaps;
  }, [hasData]);
  const snapPointsInitialIndex = 0

```

```
  return (
    <BottomSheet
      ref= { bottomSheetRef }
  index = { snapPointsInitialIndex }
  snapPoints = { snapPoints }
  style = { styles.container } >
    <View style={ styles.contentContainer }>
      <ContactsList noFavToggle data = { favoritesList } />
        </View>
        < /BottomSheet>
  );
};

const styles = StyleSheet.create({
  container: {
    borderColor: commonColors.color16,
    borderTopLeftRadius: 14,
    borderTopRightRadius: 14,
  },
  contentContainer: {
    flex: 1,
    justifyContent: 'flex-start',
  },
});

export default memo(FavoritesSheet, isEqual);



\textbf{ /src/components / FavoritesSheet / index.ts }

export { default as FavoritesSheet } from './FavoritesSheet';

\textbf{ /src/components / LoginScreen / LoginScreen.tsx }


import { Text, TextInput, TouchableOpacity, StyleSheet, View } from '
   react-native'
import React, { useEffect, useState } from 'react'
import { formatPhoneNumber, isBrazilianPhone } from '../../utils/
   deviceHelper';
import useAuth from '../../hooks/useAuth';
import { onlyNumbers } from '@brazilian-utils/brazilian-utils';
import { BRAZILIAN_PHONE_CODE } from '../../utils/constants';
import { routes } from '../../Router';

const LoginScreen = ({ navigation }: any) => {
```

```
711   const { signIn, hasVerification } = useAuth();
712   const [phone, setPhone] = useState('')
713
714   const isValid = isBrazilianPhone(`${BRAZILIAN_PHONE_CODE}${phone}`)
715
716   const onChangeNumber = (phoneNumber: string) => {
717     setPhone(formatPhoneNumber(phoneNumber))
718   }
719
720   async function signInWithPhoneNumber() {
721     signIn(`${BRAZILIAN_PHONE_CODE}${onlyNumbers(phone)}`)
722   }
723
724   useEffect(() => {
725     if (hasVerification) {
726       navigation.navigate(routes.CONFIRM_CODE)
727     }
728   }, [hasVerification])
729
730   return (
731     <View style= { styles.container } >
732     <Text style={ styles.title }> Login < /Text>
733       < TextInput
734   numberOfLines = { 1}
735   keyboardType = 'phone-pad'
736   style = { styles.input }
737   placeholder = "N mero de telefone"
738   value = { phone }
739   onChangeText = { onChangeNumber }
740     />
741     <TouchableOpacity style={ StyleSheet.compose(styles.button, isValid
          ? {} : styles.disabledButton) } onPress = { signInWithPhoneNumber
           } disabled = {!isValid
742 }>
743   <Text style={ styles.buttonText }> Sign in </Text>
744     < /TouchableOpacity>
745     < /View>
746   )
747 }
748
749 export default LoginScreen
750
751 const styles = StyleSheet.create({
752   container: {
753     flex: 1,
754     justifyContent: 'center',
755     alignItems: 'center',
```

```
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 24,
  },
  input: {
    borderWidth: 1,
    borderColor: '#ccc',
    borderRadius: 4,
    padding: 12,
    marginBottom: 16,
    width: '80%',
  },
  button: {
    backgroundColor: '#007AFF',
    borderRadius: 4,
    padding: 12,
    width: '80%',
    alignItems: 'center',
  },
  disabledButton: {
    backgroundColor: '#ECECEC'
  },
  buttonText: {
    color: '#fff',
    fontSize: 16,
    fontWeight: 'bold',
  },
});


\textbf{ /src/components / LoginScreen / index.ts }

export { default as LoginScreen } from './LoginScreen'

\textbf{ /src/components / Map / Map.tsx }


import React, { MutableRefObject, useEffect, useRef, useState } from '
    react';
import { StyleSheet, View } from 'react-native';
import MapView, { MapMarker, PROVIDER_GOOGLE, Region } from 'react-
    native-maps';
import useGeolocation from '../../hooks/useGeolocation';
import StatusBar from '../StatusBar/CustomStatusBar';
```

```
801
802 const MapScreen: React.FC = () => {
803   StatusBar.setHidden(true);
804   const ref = useRef<any>();
805   const position = useGeolocation();
806   const [region, setRegion] = useState<Region>({
807     latitude: 37.78825,
808     longitude: -122.4324,
809     latitudeDelta: 0.0922,
810     longitudeDelta: 0.0421,
811   });
812
813   useEffect(() => {
814     return () => {
815       StatusBar.setHidden(false);
816     }
817   }, [])
818
819   useEffect(() => {
820     if (position) {
821       const { latitude, longitude } = position;
822       setRegion((prevRegion) => ({
823         ...prevRegion,
824         latitude,
825         longitude,
826       }));
827     }
828   }, [position]);
829
830   return (
831     <View style= { styles.container } >
832     <MapView
833         ref={ ref }
834   provider = { PROVIDER_GOOGLE }
835   style = { styles.map }
836   initialRegion = { region }
837   customMapStyle = { googleMapsStyle }
838   region = { region }
839   showsBuildings
840   showsMyLocationButton
841   showsUserLocation
842   showsPointsOfInterest
843   showsCompass
844   zoomEnabled
845   zoomControlEnabled
846   zoomTapEnabled
847   focusable
```

```
848   loadingEnabled
849   mapType = "terrain"
850     >
851     <MapMarker
852           coordinate={ region }
853   draggable
854   calloutAnchor = {{
855     x: 2.9,
856       y: 0.8,
857           }
858 }
859 />
860   < /MapView>
861   < /View>
862   );
863 };
864
865 const styles = StyleSheet.create({
866   container: {
867     ...StyleSheet.absoluteFillObject,
868     justifyContent: 'flex-end',
869     alignItems: 'center',
870   },
871   map: {
872     ...StyleSheet.absoluteFillObject,
873   },
874 });
875
876 export default MapScreen;
877
878 const googleMapsStyle = [
879   {
880     elementType: 'geometry',
881     stylers: [
882       {
883         color: '#ebe3cd',
884       },
885     ],
886   },
887   {
888     elementType: 'labels.text.fill',
889     stylers: [
890       {
891         color: '#523735',
892       },
893     ],
894   },
```

```
895   {
896     elementType: 'labels.text.stroke',
897     stylers: [
898       {
899         color: '#f5f1e6',
900       },
901     ],
902   },
903   {
904     featureType: 'administrative',
905     elementType: 'geometry.stroke',
906     stylers: [
907       {
908         color: '#c9b2a6',
909       },
910     ],
911   },
912   {
913     featureType: 'administrative.land_parcel',
914     elementType: 'geometry.stroke',
915     stylers: [
916       {
917         color: '#dcd2be',
918       },
919     ],
920   },
921   {
922     featureType: 'administrative.land_parcel',
923     elementType: 'labels.text.fill',
924     stylers: [
925       {
926         color: '#ae9e90',
927       },
928     ],
929   },
930   {
931     featureType: 'landscape.natural',
932     elementType: 'geometry',
933     stylers: [
934       {
935         color: '#dfd2ae',
936       },
937     ],
938   },
939   {
940     featureType: 'poi',
941     elementType: 'geometry',
```

```
942       stylers: [
943         {
944           color: '#dfd2ae',
945         },
946       ],
947     },
948     {
949       featureType: 'poi',
950       elementType: 'labels.text.fill',
951       stylers: [
952         {
953           color: '#93817c',
954         },
955       ],
956     },
957     {
958       featureType: 'poi.park',
959       elementType: 'geometry.fill',
960       stylers: [
961         {
962           color: '#a5b076',
963         },
964       ],
965     },
966     {
967       featureType: 'poi.park',
968       elementType: 'labels.text.fill',
969       stylers: [
970         {
971           color: '#447530',
972         },
973       ],
974     },
975     {
976       featureType: 'road',
977       elementType: 'geometry',
978       stylers: [
979         {
980           color: '#f5f1e6',
981         },
982       ],
983     },
984     {
985       featureType: 'road.arterial',
986       elementType: 'geometry',
987       stylers: [
988         {
```

```
989         color: '#fdfcf8',
990       },
991     ],
992   },
993   {
994     featureType: 'road.highway',
995     elementType: 'geometry',
996     stylers: [
997       {
998         color: '#f8c967',
999       },
1000    ],
1001  },
1002  {
1003    featureType: 'road.highway',
1004    elementType: 'geometry.stroke',
1005    stylers: [
1006      {
1007        color: '#e9bc62',
1008      },
1009    ],
1010  },
1011  {
1012    featureType: 'road.highway.controlled_access',
1013    elementType: 'geometry',
1014    stylers: [
1015      {
1016        color: '#e98d58',
1017      },
1018    ],
1019  },
1020  {
1021    featureType: 'road.highway.controlled_access',
1022    elementType: 'geometry.stroke',
1023    stylers: [
1024      {
1025        color: '#db8555',
1026      },
1027    ],
1028  },
1029  {
1030    featureType: 'road.local',
1031    elementType: 'labels.text.fill',
1032    stylers: [
1033      {
1034        color: '#806b63',
1035      },
```

```
    ],
  },
]


\textbf{ /src/components / Map / index.ts }

export { default as MapView } from './Map'

\textbf{ /src/components / MapScreen / MapScreen.tsx }


import React, { useEffect } from 'react';
import { FavoritesSheet } from '../FavoritesSheet';
import { MapView } from '../Map';
import { ScreenContainer } from '../Shared/ScreenContainer';
import { useContacts } from '../../hooks/useContacts';

const MapScreen = () => {
  const { favorites } = useContacts();

  return (
    <ScreenContainer noPadding >
    <MapView />
      {
    favorites?.length > 0 && <FavoritesSheet />}
      < /ScreenContainer>
  );
};

export default MapScreen;




\textbf{ /src/components / MapScreen / index.ts }

export { default as MapScreen } from './MapScreen';

\textbf{ /src/components / ReportButton / ReportButton.tsx }


import { TouchableHighlight, StyleSheet } from 'react-native'
import React, { useState } from 'react'
import { ShieldExclamationIcon } from 'react-native-heroicons/outline'
import { commonColors } from '../../theme/styles'
import { useContacts } from '../../hooks/useContacts'
```

```
import { REPORT_BUTTON_POSITION, REPORT_BUTTON_SIZE } from '../../utils/
    constants'
import { sendSMS } from '../../services/messagingService'
import { getLocationDetails } from '../../services/locationService'
import useGeolocation from '../../hooks/useGeolocation'

const ReportButton = () => {
  const { favorites } = useContacts();
  const { latitude, longitude } = useGeolocation();

  const sendMessage = async () => {
    const favoritesWithAppInstalled = favorites.map((f) => {
      //return collections.users.doc(`${c}`)
    })
    const message = "your message"
    if (favoritesWithAppInstalled.length > 0) {
      // await firebaseAdmin.messaging().sendMulticast({

      // })
    }
    const locationName = await getLocationDetails(latitude, longitude);
    console.log(favorites)
    sendSMS(favorites, `Preciso de sua aten   o! Estou cituado em: ${
        locationName}`)
  }
  return (
    <TouchableHighlight onPress= { sendMessage } style = { styles.button
        } >
      <ShieldExclamationIcon color={ 'white' } />
        < /TouchableHighlight>
  )
}

export default ReportButton

const styles = StyleSheet.create({
  button: {
    position: 'absolute',
    alignItems: 'center',
    alignSelf: 'center',
    justifyContent: 'center',
    padding: 5,
    width: REPORT_BUTTON_SIZE,
    height: REPORT_BUTTON_SIZE,
    zIndex: 9999,
    bottom: 40,
    left: REPORT_BUTTON_POSITION,
```

```
    borderRadius: 100,
    backgroundColor: commonColors.color9,
  }
})


\textbf{ /src/components / ReportButton / index.ts }

export { default as ReportButton } from './ReportButton'

\textbf{ /src/components / SafeAreaWrapper / SafeAreaWrapper.tsx }


import React, { PropsWithChildren } from 'react';
import { View, StyleSheet } from 'react-native';
import useTheme from '../../hooks/useTheme';

type Props = PropsWithChildren & {
  topPadding: number;
};

const SafeAreaWrapper = ({ topPadding, children }: Props) => {
  const { theme } = useTheme();
  return (
    <View
      style= {
      StyleSheet.flatten([
        styles.wrapper,
        {
          paddingTop: topPadding,
          backgroundColor: theme.screenBgColor
        },
      ])
    } >
    { children }
    < /View>
  );
};

export default SafeAreaWrapper;

const styles = StyleSheet.create({
  wrapper: {
    flex: 1,
  },
});

```

```
1174
1175
1176 \textbf{ /src/components / SafeAreaWrapper / index.ts }
1177
1178 export { default as SafeAreaWrapper } from './SafeAreaWrapper';
1179
1180 \textbf{ /src/components / SettingsScreen / SettingsScreen.tsx }
1181
1182
1183 import React, { memo, useCallback, useState } from 'react';
1184 import { StyleSheet, View } from 'react-native';
1185 import SettingsItem from './SettingsItem';
1186 import routes from '../../Router/routes';
1187 import { ScreenTitle } from '../Shared/ScreenTitle';
1188 import { TAB_NAMES } from '../../utils/constants';
1189 import useAuth from '../../hooks/useAuth';
1190
1191
1192 function SettingsScreen({ navigation }: any) {
1193   const { signOut } = useAuth();
1194
1195   const logOut = () => {
1196     signOut()
1197     navigation.navigate(routes.LOGIN_STACK);
1198   }
1199
1200   const goToContactsScreen = useCallback(() => {
1201     navigation.navigate(routes.CONTACTS);
1202   }, [navigation]);
1203
1204   return (
1205     <View style= { styles.container } >
1206     <ScreenTitle>{ TAB_NAMES.SETTINGS } < /ScreenTitle>
1207     < View style = { styles.wrapper } >
1208       <SettingsItem title="Contatos" onPress = { goToContactsScreen } />
1209         <SettingsItem hideTopLine title = "Sair" onPress = { logOut } />
1210           </View>
1211           < /View>
1212   );
1213 }
1214
1215 export default memo(SettingsScreen);
1216
1217 const styles = StyleSheet.create({
1218   container: {
1219     flex: 1,
1220     padding: 32,
```

```
  },
  wrapper: { marginTop: 40, flex: 1, },
});


\textbf{ /src/components / SettingsScreen / SettingsItem.tsx }


import React from 'react';
import { Pressable, StyleSheet, Text, View } from 'react-native';
import { ChevronRightIcon } from 'react-native-heroicons/solid';
import useTheme from '../../hooks/useTheme';
import { Divider } from '../Shared/Divider';

const SettingsItem = ({ title, component = null, onPress, hideTopLine =
    false }: any) => {
  const { theme } = useTheme();

  return (
    <Pressable onPress= { onPress } >
    {!hideTopLine && <Divider />}
      < View style = { styles.wrapper } >
        <Text style={ StyleSheet.flatten([styles.title, { color: theme.
            text }]) }>
          { title }
          < /Text>
  {
    component ? (
      component
    ) : (
      <ChevronRightIcon size= { 22} color = { theme.text } />
        )}
</View>
  < Divider />
  </Pressable>
  );
};

export default SettingsItem;

const styles = StyleSheet.create({
  wrapper: {
    width: '100%',
    paddingVertical: 16,
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
```

```
1266   },
1267   title: {
1268     fontWeight: '600',
1269     fontSize: 16,
1270   },
1271 });
1272
1273
1274
1275 \textbf{ /src/components / SettingsScreen / index.ts }
1276
1277 export { default as SettingsScreen } from './SettingsScreen';
1278 export { default as SettingsItem } from './SettingsItem';
1279
1280 \textbf{ /src/components / Shared / ContactItem / ContactItem.tsx }
1281
1282
1283 import { View, Text, StyleSheet, Image, TouchableWithoutFeedback } from
       'react-native'
1284 import React, { memo } from 'react'
1285 import { Contact, openExistingContact } from 'react-native-contacts'
1286 import { StarIcon, UserCircleIcon } from 'react-native-heroicons/solid';
1287 import { StarIcon as StarIconOutlined } from 'react-native-heroicons/
       outline';
1288 import { commonColors } from '../../../theme/styles'
1289 import useTheme from '../../../hooks/useTheme';
1290 import { useContacts } from '../../../hooks/useContacts';
1291 import isEqual from 'react-fast-compare';
1292
1293 interface Props {
1294   contact: Contact;
1295   noFavToggle?: boolean;
1296 }
1297
1298 const ContactItem = ({ contact, noFavToggle = false }: Props) => {
1299   const { favorites: favoritesList, addToFavorites, removeFromFavorites
         } = useContacts();
1300   const { theme } = useTheme()
1301
1302   const toggleFavorite = (contact: Contact) => {
1303     const isFavorite = favoritesList.find((c) => c.phoneNumbers ===
         contact.phoneNumbers)
1304     if (isFavorite) {
1305       removeFromFavorites(contact)
1306     } else {
1307       addToFavorites(contact)
1308     }
```

```
1309   }
1310
1311   const handleFavorite = () => noFavToggle ? null : toggleFavorite(
           contact)
1312
1313   const handleContact = () => noFavToggle ? null : openExistingContact(
           contact)
1314
1315   return (
1316     <TouchableWithoutFeedback onPress= { handleContact } >
1317     <View style={ styles.container }>
1318       <View style={ StyleSheet.flatten([styles.container, styles.info])
              }>
1319         { contact.hasThumbnail ? <Image source={ { uri: contact.
              thumbnailPath, width: 22.5, height: 22.5 } } borderRadius = {
               100} /> : <UserCircleIcon color={ theme.text } />}
1320           < Text style = {
1321             StyleSheet.flatten([
1322               styles.name,
1323               { color: theme.text }
1324             ])
1325           } > { contact.givenName } { contact.familyName } </Text>
1326             < /View>
1327   {
1328     noFavToggle ? null :
1329       <TouchableWithoutFeedback onPress={ handleFavorite }>
1330         { favoritesList.includes(contact) ? <StarIcon color={
              commonColors.color25 } /> : <StarIconOutlined color={theme.
              text} / >}
1331   </TouchableWithoutFeedback>
1332 }
1333 </View>
1334   < /TouchableWithoutFeedback>
1335   )
1336 }
1337
1338 export default memo(ContactItem, isEqual)
1339
1340 const styles = StyleSheet.create({
1341   container: {
1342     flexDirection: `row`,
1343     borderBottomWidth: 1,
1344     paddingVertical: 6,
1345     marginHorizontal: 10,
1346     borderColor: commonColors.color12,
1347     justifyContent: 'space-between',
1348     alignItems: 'center'
```

```
1349   },
1350   info: {
1351     justifyContent: 'flex-start',
1352     borderBottomWidth: 0,
1353   },
1354   name: {
1355     marginLeft: 8,
1356   }
1357 })
1358
1359
1360 \textbf{ /src/components / Shared / ContactItem / index.ts }
1361
1362 export { default as ContactItem } from './ContactItem'
1363
1364 \textbf{ /src/components / Shared / ContactsList / index.ts }
1365
1366 export { default as ContactsList } from './ContactsList'
1367
1368 \textbf{ /src/components / Shared / ContactsList / ContactsList.tsx }
1369
1370
1371 import { Text, StyleSheet } from 'react-native'
1372 import React, { memo } from 'react'
1373 import { AnimatedFlashList } from '@shopify/flash-list'
1374 import { Contact } from 'react-native-contacts'
1375 import { ContactItem } from '../ContactItem';
1376 import usePermissions from '../../../hooks/usePermissions';
1377 import { ViewAllButton } from '../../ViewAllButton';
1378 import { getContactName } from '../../../utils/constants'
1379 import isEqual from 'react-fast-compare';
1380
1381 interface Props {
1382   data: Contact[] | null;
1383   filter?: string;
1384   noFavToggle?: boolean;
1385 }
1386
1387 const ContactsList = ({ data, filter = '', noFavToggle = false }: Props)
       => {
1388   const { contactsAllowed } = usePermissions()
1389   const count = data?.length ?? 0
1390   let contactsList = data?.filter((c) => getContactName(c).toLowerCase()
       .includes(filter.toLowerCase()));
1391   contactsList = contactsList?.sort((a, b) => {
1392     const nameA = getContactName(a);
1393     const nameB = getContactName(b);
```

```
1394
1395      if (nameA < nameB) {
1396        return -1;
1397      }
1398
1399      if (nameA > nameB) {
1400        return 1;
1401      }
1402
1403      return 0;
1404    })
1405
1406    return (
1407      <AnimatedFlashList
1408        data= { contactsList }
1409    renderItem = {({ item }) => <ContactItem noFavToggle={ noFavToggle }
          contact = { item } />}
1410  estimatedItemSize = { count > 0 ? count : 10}
1411  ListFooterComponent = { noFavToggle? ViewAllButton: null }
1412  ListEmptyComponent = {() => contactsAllowed ? (
1413    <Text style= { styles.empty } > Nenhum contato favorito < /Text>
1414        ) : <Text style={ styles.empty }> O app n o possui acesso aos
             seus contatos < /Text>}
1415    />
1416    )
1417  }
1418
1419  export default memo(ContactsList, isEqual)
1420
1421  const styles = StyleSheet.create({
1422    empty: {
1423      paddingHorizontal: 12,
1424      textAlign: 'center',
1425      fontWeight: '500',
1426    }
1427  });
1428
1429
1430  \textbf{ /src/components / Shared / ScreenContainer / ScreenContainer.
       tsx }
1431
1432
1433  import { View, StyleSheet, StyleProp } from 'react-native';
1434  import React, { PropsWithChildren } from 'react';
1435  import useTheme from '../../../hooks/useTheme';
1436  import { SafeAreaWrapper } from '../../SafeAreaWrapper';
1437  import { useSafeAreaInsets } from 'react-native-safe-area-context';
```

```
import { getNotchPadding } from '../../../utils/deviceHelper';

type Props = PropsWithChildren & {
  style?: StyleProp<any>;
  noPadding?: boolean
};

const ScreenContainer = ({ children, style, noPadding }: Props) => {
  const { theme } = useTheme();
  const { top } = useSafeAreaInsets();

  return (
    <SafeAreaWrapper topPadding= { noPadding? 0: getNotchPadding(top) }
       >
    <View
        style={
    StyleSheet.flatten([
      styles.container,
      !noPadding ? styles.screenPadding : null,
      style,
      { backgroundColor: theme.transparent },
    ])
  }>
    { children }
    < /View>
    < /SafeAreaWrapper>
  );
};

export default ScreenContainer;

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  screenPadding: {
    paddingHorizontal: 16,
    paddingVertical: 32,
  }
});


\textbf{ /src/components / Shared / ScreenContainer / index.ts }

export { default as ScreenContainer } from './ScreenContainer';

\textbf{ /src/components / Shared / ScreenTitle / index.ts }
```

```
1484
1485 export { default as ScreenTitle } from './ScreenTitle';
1486
1487 \textbf{ /src/components / Shared / ScreenTitle / ScreenTitle.tsx }
1488
1489
1490 import { StyleSheet, Text } from 'react-native';
1491 import React from 'react';
1492 import useTheme from '../../../hooks/useTheme';
1493
1494 const ScreenTitle = ({ children }: { children: string }) => {
1495   const { theme } = useTheme();
1496   return (
1497     <Text style= { StyleSheet.flatten([styles.title, { color: theme.text
          }]) } >
1498     { children }
1499     < /Text>
1500   );
1501 };
1502
1503 export default ScreenTitle;
1504
1505 const styles = StyleSheet.create({
1506   title: {
1507     fontSize: 24,
1508     fontWeight: '600',
1509   },
1510 });
1511
1512
1513
1514 \textbf{ /src/components / Shared / Common.style.tsx }
1515
1516
1517 import React from 'react';
1518 import { View } from 'react-native';
1519 import { BOTTOM_TAB_HEIGHT } from '../../utils/constants';
1520
1521 export const tabNavigatorStyle: any = {
1522   height: BOTTOM_TAB_HEIGHT,
1523   paddingTop: 14,
1524   borderTopWidth: 1,
1525   position: 'relative',
1526   elevation: 0,
1527   shadowOffset: {
1528     width: 0,
1529     height: 0,
```

```
  },
};

export const tabBarStyles: any = {
  tabBarLabelStyle: { fontWeight: '700', marginTop: 8 },
};

export const Empty = () => <View />;



\textbf{ /src/components / StatusBar / CustomStatusBar.tsx }

import React from 'react';
import { Platform, StatusBar as StatusBarView, StatusBarAnimation,
   ColorValue } from 'react-native';

class StatusBar extends StatusBarView {

  static setBackgroundColor(color: ColorValue, animated?: boolean) {
    StatusBarView.setBackgroundColor(color, animated)
  }

  static setTranslucent(translucent: boolean) {
    StatusBarView.setTranslucent(translucent)
  }

  static setHidden(hidden: boolean, animation?: StatusBarAnimation) {
    // if (Platform.OS === 'ios') {
    StatusBarView.setHidden(hidden, animation)
    // }
  }

}

export default StatusBar;



\textbf{ /src/components / ViewAllButton / index.ts }

export { default as ViewAllButton } from './ViewAllButton'

\textbf{ /src/components / ViewAllButton / ViewAllButton.tsx }


import { View, Text, TouchableWithoutFeedback, StyleSheet } from 'react-
```

```
    native'
import React, { memo } from 'react'
import isEqual from 'react-fast-compare'
import { useNavigation } from '@react-navigation/native'
import { routes } from '../../Router'
import { commonColors } from '../../theme/styles'

const ViewAllButton = () => {
  const { navigate } = useNavigation();
  const handleContacts = () => navigate(routes.SETTINGS_ROUTER, { screen
     : routes.CONTACTS })
  return (
    <TouchableWithoutFeedback onPress= { handleContacts } >
    <View style={ styles.wrapper }>
      <Text style={ styles.label }> Ver todos < /Text>
        < /View>
        < /TouchableWithoutFeedback>
  )
}

export default memo(ViewAllButton, isEqual)

const styles = StyleSheet.create({
  wrapper: {
    alignItems: `center`,
    marginVertical: 12,
  },
  label: {
    color: commonColors.color14,
    fontStyle: 'italic'
  }
})



\textbf{ /src/contexts / ContactsContext.tsx }


import React, { createContext } from 'react';
import { useEffect, useState } from 'react';
import Contacts, { Contact } from 'react-native-contacts';

interface ContactsContextValue {
  contacts: Contact[] | null;
  favorites: Contact[];
  loading: boolean;
  error: Error | null;
```

```
1621   addToFavorites: (contact: Contact) => void;
1622   removeFromFavorites: (contact: Contact) => void;
1623 }
1624
1625 export const ContactsContext = createContext<ContactsContextValue>({
1626   contacts: null,
1627   favorites: [],
1628   loading: true,
1629   error: null,
1630   addToFavorites: () => { },
1631   removeFromFavorites: () => { },
1632 });
1633
1634 interface PhoneContactsProviderProps {
1635   children: React.ReactNode;
1636 }
1637
1638 export const ContactsProvider = ({ children }:
       PhoneContactsProviderProps) => {
1639   const [contacts, setContacts] = useState<Contact[]>([]);
1640   const [favorites, setFavorites] = useState<Contact[]>([]);
1641   const [loading, setLoading] = useState(true);
1642   const [error, setError] = useState<Error | null>(null);
1643
1644   useEffect(() => {
1645     const loadContacts = async () => {
1646       try {
1647         const result = await Contacts.getAll();
1648         setContacts([...result]);
1649         setLoading(false);
1650       } catch (error: any) {
1651         setError(error);
1652         setLoading(false);
1653       }
1654     };
1655
1656     loadContacts();
1657   }, []);
1658
1659   const addToFavorites = (contact: Contact) => {
1660     setFavorites((prevFavorites) => [...prevFavorites, contact]);
1661   };
1662
1663   const removeFromFavorites = (contact: Contact) => {
1664     setFavorites((prevFavorites) => prevFavorites.filter((c) => c.
         phoneNumbers !== contact.phoneNumbers));
1665   };
```

```
1666
1667   const value: ContactsContextValue = {
1668     contacts,
1669     favorites,
1670     loading,
1671     error,
1672     addToFavorites,
1673     removeFromFavorites,
1674   };
1675
1676   return (
1677     <ContactsContext.Provider value={ value } >
1678     { children }
1679     < /ContactsContext.Provider>
1680   );
1681 };
1682
1683
1684 \textbf{ /src/contexts / PermissionsContext.tsx }
1685
1686
1687 import React, {
1688   createContext,
1689   PropsWithChildren,
1690   useCallback,
1691   useEffect,
1692   useState,
1693 } from 'react';
1694 import {
1695   request,
1696   PERMISSIONS,
1697   requestLocationAccuracy,
1698   requestNotifications,
1699   PermissionStatus
1700 } from 'react-native-permissions';
1701 import { isIOS } from '../utils/constants';
1702 import { PermissionsResponse } from '../utils/enums';
1703
1704 export const PermissionsContext = createContext<{
1705   contactsAllowed: boolean;
1706   locationAllowed: boolean;
1707   notificationsAllowed: boolean;
1708   //accuracyAllowed: boolean;
1709 }>({
1710   contactsAllowed: false,
1711   locationAllowed: false,
1712   notificationsAllowed: false,
```

```
1713   //accuracyAllowed: false,
1714 });
1715
1716 const PermissionsProvider = ({ children }: PropsWithChildren) => {
1717   const [contactsAllowed, setContactsAllowed] = useState(false);
1718   const [locationAllowed, setLocationAllowed] = useState(false);
1719   const [notificationsAllowed, setNotificationsAllowed] = useState(false
          );
1720   //const [accuracyAllowed, setAccuracyAllowed] = useState(false);
1721
1722   const isAllowed = (perm: PermissionStatus | PermissionsResponse |
          undefined) => {
1723     switch (perm) {
1724       case PermissionsResponse.GRANTED:
1725       case PermissionsResponse.LIMITED:
1726         return true;
1727       case PermissionsResponse.BLOCKED:
1728       case PermissionsResponse.DENIED:
1729         return false
1730       default:
1731         return undefined
1732     }
1733   }
1734
1735   const askPermissions = useCallback(async () => {
1736     let location, contacts;
1737     if (isIOS) {
1738       location = await request(PERMISSIONS.IOS.LOCATION_ALWAYS);
1739       contacts = await request(PERMISSIONS.IOS.CONTACTS);
1740     } else {
1741       contacts = await request(PERMISSIONS.ANDROID.READ_CONTACTS);
1742       const location1 = await request(PERMISSIONS.ANDROID.
              ACCESS_BACKGROUND_LOCATION);
1743       const location2 = await request(PERMISSIONS.ANDROID.
              ACCESS_FINE_LOCATION);
1744       const location3 = await request(PERMISSIONS.ANDROID.
              ACCESS_COARSE_LOCATION);
1745       if (isAllowed(location1) || isAllowed(location2) || isAllowed(
              location3)) {
1746         location = PermissionsResponse.GRANTED;
1747       }
1748     }
1749     setContactsAllowed(!!isAllowed(contacts))
1750     setLocationAllowed(!!isAllowed(location))
1751     const notifications = await requestNotifications([
1752       'alert',
1753       'badge',
```

```
      'carPlay',
      'criticalAlert',
      'sound',
    ])
    await requestLocationAccuracy({
      purposeKey: 'BETTER-USAGE',
    })
    setNotificationsAllowed(!!isAllowed(notifications.status))
  }, []);

  useEffect(() => {
    askPermissions();
  }, [askPermissions]);

  return (
    <PermissionsContext.Provider value= {{
    contactsAllowed,
      locationAllowed,
      notificationsAllowed,
      // accuracyAllowed,
    }
}>
  { children }
  < /PermissionsContext.Provider>
  );
};
export default PermissionsProvider;



\textbf{ /src/contexts / ThemeContext.tsx }


import React, {
  createContext,
  PropsWithChildren,
  useCallback,
  useState,
} from 'react';
import { THEMES } from '../utils/constants';
import theme from '../theme/styles';
import { useMemo } from 'react';
import { Platform } from 'react-native';
import { SafeAreaProvider } from 'react-native-safe-area-context';
import StatusBar from '../components/StatusBar/CustomStatusBar';

export const ThemeContext = createContext<{
```

```
1801   theme: { [key: string]: string };
1802   isDarkTheme: boolean;
1803   isLoading: boolean;
1804   toggleTheme: () => void;
1805 }>({
1806   theme: {},
1807   isDarkTheme: false,
1808   isLoading: false,
1809   toggleTheme: () => null,
1810 });
1811
1812 const ThemeProvider = ({ children }: PropsWithChildren) => {
1813   const [userTheme, setUserTheme] = useState(THEMES.LIGHT);
1814   const [colorScheme, setColorScheme] = useState(theme.colors.light);
1815   const [isLoading, setIsLoading] = useState(false);
1816   const isDarkTheme = userTheme !== THEMES.LIGHT;
1817   const isIOS = useMemo(() => Platform.OS === 'ios', []);
1818
1819   const toggleTheme = useCallback(() => {
1820     console.log('THEME::', isDarkTheme, userTheme);
1821     setIsLoading(true);
1822     setUserTheme(isDarkTheme ? THEMES.LIGHT : THEMES.DARK);
1823     setColorScheme(isDarkTheme ? theme.colors.light : theme.colors.dark)
          ;
1824     setIsLoading(false);
1825   }, [isDarkTheme, userTheme, setUserTheme, setColorScheme, setIsLoading
       ]);
1826
1827   return (
1828     <ThemeContext.Provider
1829       value= {{ theme: colorScheme, isDarkTheme, isLoading, toggleTheme
            }
1830 }>
1831   { children }
1832   < /ThemeContext.Provider>
1833   );
1834 };
1835
1836 export default ThemeProvider;
1837
1838
1839
1840 \textbf{ /src/contexts / UserContext.tsx }
1841
1842
1843 import { FirebaseAuthTypes } from '@react-native-firebase/auth';
1844 import React, { createContext, PropsWithChildren, useEffect, useState }
```

```
   from 'react';
1845 import { getAuth } from '../services/firebaseSetup';
1846 import { USER_UUID_STORAGE_KEY } from '../utils/constants';
1847 import { deletePersistedItem, setPersistentItem } from '../utils/storage
   ';
1848 import { UserContextType } from '../types/contexts';
1849
1850
1851
1852 export const UserContext = createContext<UserContextType>({
1853   confirmCode: () => null,
1854   signIn: () => null,
1855   signOut: () => null,
1856   hasVerification: false
1857 });
1858
1859 const UserProvider = ({ children }: PropsWithChildren) => {
1860   const [user, setUser] = useState<FirebaseAuthTypes.User | null>(null);
1861   const [verification, setVerification] = useState<FirebaseAuthTypes.
      ConfirmationResult>();
1862
1863   const handleSignIn = async (phoneNumber: string) => {
1864     console.log(`PHONE::`, phoneNumber)
1865     try {
1866       const confirmation = await getAuth().signInWithPhoneNumber(
          phoneNumber);
1867       setVerification(confirmation);
1868     } catch (error) {
1869       console.log('Error sending verification code:', error);
1870     }
1871   };
1872
1873   const handleVerifyCode = async (code: string) => {
1874     try {
1875       if (!!verification) {
1876         await verification.confirm(code);
1877       }
1878     } catch (error) {
1879       console.log('Error verifying code:', error);
1880     }
1881   };
1882
1883   const updateUser = async (uuid: string) => {
1884     await setPersistentItem(USER_UUID_STORAGE_KEY, uuid)
1885   }
1886
1887   const signOut = async () => {
```

```
    try {
      getAuth().signOut();
    } catch (error) {
      // do nothing
    }
    await deletePersistedItem(USER_UUID_STORAGE_KEY);
    setUser(null);
    setVerification(undefined);
  }

  useEffect(() => {
    const subscriber = getAuth().onAuthStateChanged((user) => {
      if (user) {
        setUser(user)
      } else {
        setUser(null);
      }
    });
    return subscriber; // unsubscribe on unmount
  }, []);

  useEffect(() => {
    user && updateUser(user.uid)
  }, [user])

  return <UserContext.Provider value={
    {
      user,
        hasVerification: !!verification,
          confirmCode: handleVerifyCode,
            signIn: handleSignIn,
              signOut
    }
  }>
    { children }
    < /UserContext.Provider>;
};

export default UserProvider;



/src/hooks / useAlertsSubscription.ts


import { useEffect } from "react";
import { Alert } from "react-native";
```

```
import messaging, { FirebaseMessagingTypes } from '@react-native-
    firebase/messaging';


function displayMessage(remoteMessage: FirebaseMessagingTypes.
    RemoteMessage) {
  let message_body = remoteMessage?.notification?.body;

  // Get the message title
  let message_title = remoteMessage?.notification?.title ?? "A ten   o";

  // Show an alert to the user
  Alert.alert(message_title, message_body);
}

messaging().setAutoInitEnabled(true)

// Register background handler
messaging().setBackgroundMessageHandler(async remoteMessage => {
  displayMessage(remoteMessage);
});

const useAlertsSubscription = () => {

  useEffect(() => {
    const subscribe = messaging().onMessage(async remoteMessage => {

      // Get the message body
      displayMessage(remoteMessage);
    });

    return subscribe;
  }, []);
}

export default useAlertsSubscription



  / src / hooks / useAuth.ts


import { useContext } from "react"
import { UserContext } from "../contexts/UserContext"

const useAuth = () => useContext(UserContext)

```

```
1980 export default useAuth
1981
1982
1983   / src / hooks / useClimaticDangers.ts
1984
1985
1986 import { useEffect, useState } from 'react';
1987 import Toast from 'react-native-toast-message';
1988 import { HEAVY_RAIN_WEATHER_CODES, OPEN_WEATHER_KEY } from '../utils/
       constants';
1989
1990 const useClimateDanger = (latitude: number, longitude: number): string
       => {
1991   const [dangerMessage, setDangerMessage] = useState('');
1992
1993   useEffect(() => {
1994     const fetchWeather = async () => {
1995       try {
1996         const response = await fetch(`https://api.openweathermap.org/
              data/2.5/weather?lat=${latitude}&lon=${longitude}&appid=${
              OPEN_WEATHER_KEY}`);
1997         const data = await response.json();
1998         const weatherCode = data.weather[0].id;
1999
2000         if (HEAVY_RAIN_WEATHER_CODES.includes(weatherCode)) {
2001           const cityName = data.name;
2002           setDangerMessage(`Aten   o: Possibilidade de chuva forte em $
                 {cityName}.`);
2003         } else {
2004           setDangerMessage('');
2005         }
2006       } catch (error) {
2007         console.error(error);
2008       }
2009     };
2010
2011     fetchWeather();
2012   }, [latitude, longitude]);
2013
2014   useEffect(() => {
2015     if (dangerMessage) {
2016       Toast.show({
2017         text1: 'Clima perigoso',
2018         text2: dangerMessage,
2019         type: 'warning',
2020         position: 'bottom',
2021         visibilityTime: 5000,
```

```
      });
    }
  }, [dangerMessage]);

  return dangerMessage;
};

export { useClimateDanger };



/src/hooks / useCloudMessaging.ts


import { useCallback, useState, useEffect } from "react";
import {
  getFcmToken,
  handleTokenRefresh,
  listenToBackgroundMessages,
  listenToMessages,
  requestUserPermission,
  updateFcmToken,
} from "../services/messagingService";
import usePermissions from "./usePermissions";

const useCloudMessaging = () => {
  const { notificationsAllowed } =
    usePermissions();
  const [granted, setGranted] = useState<boolean>(false);

  const requestPermission = useCallback(async () => {
    try {
      const authorized = await requestUserPermission();
      setGranted(authorized);
      if (authorized) {
        const fcmToken = await getFcmToken();
        fcmToken && (await updateFcmToken(fcmToken));
      }
    } catch (error) {
      // Ignore err
    }
  }, [setGranted]);

  const listenToFCM = useCallback(async () => {
    await handleTokenRefresh();
  }, []);

```

```
  useEffect(() => {
    if (granted) {
      listenToFCM();
    }
  }, [granted]);

  useEffect(() => {
    if (notificationsAllowed) {
      listenToBackgroundMessages();
      listenToMessages();
    } else {
      requestPermission();
    }
  }, [notificationsAllowed]);
};

export default useCloudMessaging;



/src/hooks / useContacts.ts


import { useContext } from "react";
import { ContactsContext } from "../contexts/ContactsContext";

export const useContacts = () => useContext(ContactsContext);


/src/hooks / useDidMount.ts


import { useEffect, useState } from 'react';

/**
 * Check if a component was mounted and force a rerender
 */
export function useDidMount(): boolean {
  const [isMounted, setIsMounted] = useState(false);
  useEffect(() => {
    setIsMounted(true);
  }, []);
  return isMounted;
}
```

```
/src/hooks / useGeolocation.ts


import { useState, useEffect } from 'react';
import Geolocation from '@react-native-community/geolocation';

export interface GeolocationData {
  latitude: number;
  longitude: number;
}

const useGeolocation = (): GeolocationData => {
  const [location, setLocation] = useState<GeolocationData>({
    latitude: 0,
    longitude: 0,
  });

  useEffect(() => {
    const watcher = Geolocation.watchPosition(
      position => {
        const { latitude, longitude } = position.coords;
        setLocation({ latitude, longitude });
      },
      error => console.log(error),
      { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }
    );

    return () => Geolocation.clearWatch(watcher);
  }, []);

  return location;
};

export default useGeolocation;


/src/hooks / usePermissions.ts


import { useContext } from "react";
import { PermissionsContext } from "../contexts/PermissionsContext";

const usePermissions = () => {
  const perms = useContext(PermissionsContext)
  return { ...perms }
};

```

```
export default usePermissions;



/src/hooks / useTheme.ts


import { useContext } from 'react';
import { ThemeContext } from '../contexts/ThemeContext';

const useTheme = () => useContext(ThemeContext);

export default useTheme;



/src/navigation / index.ts


export { default as BottomTabsNavigator } from './TabNavigator';



/src/navigation / LoginStack.ts


import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { routes } from '../Router';
import { LoginScreen } from '../components/LoginScreen';
import { CodeVerification } from '../components/CodeVerification';

const Stack = createStackNavigator();

const LoginStack = () => {
  return (
    <Stack.Navigator
      screenOptions= {{ header: () => null }
}
initialRouteName = { routes.LOGIN_SCREEN } >
  <Stack.Screen name={ routes.LOGIN_SCREEN } component = { LoginScreen }
      />
    <Stack.Screen name={ routes.CONFIRM_CODE } component = {
       CodeVerification } />
      </Stack.Navigator>
  );
};
```

```
2208
2209 export default LoginStack;
2210
2211
2212
2213 /src/navigation / SettingsNavigator.ts
2214
2215
2216 import React from 'react';
2217 import { createStackNavigator } from '@react-navigation/stack';
2218 import { SettingsScreen } from '../components/SettingsScreen';
2219 import { routes } from '../Router';
2220 import { ContactsScreen } from '../components/ContactsScreen';
2221
2222 const Stack = createStackNavigator();
2223
2224 const SettingsNavigator = () => {
2225   return (
2226     <Stack.Navigator
2227       screenOptions= {{ header: () => null }
2228 }
2229 initialRouteName = { routes.SETTINGS_SCREEN } >
2230   <Stack.Screen name={ routes.SETTINGS_SCREEN } component = {
          SettingsScreen } />
2231     <Stack.Screen name={ routes.CONTACTS } component = { ContactsScreen
          } />
2232       </Stack.Navigator>
2233   );
2234 };
2235
2236 export default SettingsNavigator;
2237
2238
2239
2240 /src/navigation / TabNavigator.ts
2241
2242
2243 import React, { useEffect } from 'react';
2244 import { Pressable } from 'react-native';
2245 import { createBottomTabNavigator } from '@react-navigation/bottom-tabs'
       ;
2246 import { getFocusedRouteNameFromRoute } from '@react-navigation/native';
2247 import { useSafeAreaInsets } from 'react-native-safe-area-context';
2248 import {
2249   CogIcon as SettingsIcon,
2250   ClockIcon as HistoryIcon,
2251   MapIcon,
```

```
} from 'react-native-heroicons/outline';
import useTheme from '../hooks/useTheme';
import { COMMON_ICON_SIZE, TAB_NAMES } from '../utils/constants';
import { routes } from '../Router';
import {
  disableHardwareBackButton,
  removeDisableHardwareBackButtonListener,
} from '../utils/routesHelper';
import {
  Empty,
  tabBarStyles,
  tabNavigatorStyle,
} from '../components/Shared/Common.style';
import SettingsNavigator from './SettingsNavigator';
import { MapScreen } from '../components/MapScreen';
import { ReportButton } from '../components/ReportButton';

const Tab = createBottomTabNavigator();

const BottomTabsNavigator = ({ route }: any) => {
  const currentTabRoute = getFocusedRouteNameFromRoute(route);
  const { theme } = useTheme();
  const { bottom } = useSafeAreaInsets();

  useEffect(() => {
    disableHardwareBackButton();
    return () => {
      removeDisableHardwareBackButtonListener();
    };
  }, []);

  const getTabTheme = (tab: string) => currentTabRoute === tab || tab
     === routes.MAP && !currentTabRoute ? theme.selected : theme.text

  return (
    <>
    <ReportButton />
    < Tab.Navigator
        initialRouteName = { routes.MAP }
  screenOptions = {{
    lazy: true,
      tabBarStyle: {
            ...tabNavigatorStyle,
  backgroundColor: theme.bgColor,
    borderTopColor: theme.color16,
      paddingBottom: bottom > 0 ? 25 : 10,
          },
```

```
tabBarShowLabel: true,
  headerShown: false,
    tabBarActiveTintColor: theme.selected,
      tabBarLabelStyle: tabBarStyles.selected,
        }}>
  <Tab.Screen
          name={ routes.MAP }
component = { MapScreen }
options = {{
  headerShown: false,
    unmountOnBlur: false,
      tabBarLabel: TAB_NAMES.MAP,
        tabBarIcon: () => (
          <MapIcon
                size= { COMMON_ICON_SIZE }
  color = {
    getTabTheme(routes.MAP)
  }
    />
            ),
  tabBarButton: props => (
    <Pressable { ...props } accessibilityLabel = "Button_Settings" />
            ),
}}
/>
  < Tab.Screen
name = { routes.SETTINGS_ROUTER }
component = { SettingsNavigator }
options = {{
  headerShown: false,
    unmountOnBlur: true,
      tabBarLabel: TAB_NAMES.SETTINGS,
        tabBarIcon: () => (
          <SettingsIcon
                size= { COMMON_ICON_SIZE }
  color = {
    getTabTheme(routes.SETTINGS_ROUTER)
  }
    />
            ),
  tabBarButton: props => (
    <Pressable { ...props } accessibilityLabel = "Button_Settings" />
            ),
}}
/>
  < /Tab.Navigator></ >
  );
```

```
};

export default BottomTabsNavigator;



/src/Router / index.ts


export { default as Router } from './Router';
export { default as routes } from './routes';



/src/Router / Router.tsx


import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { useSafeAreaInsets } from 'react-native-safe-area-context';
import { BottomTabsNavigator } from '../navigation';
import useAuth from '../hooks/useAuth';
import LoginStack from '../navigation/LoginStack';
import { SafeAreaWrapper } from '../components/SafeAreaWrapper';

const Stack = createStackNavigator();

const Router = () => {
  const { user } = useAuth();
  const { top } = useSafeAreaInsets();
  return (
    <SafeAreaWrapper topPadding= { top } >
    <NavigationContainer>
    <Stack.Navigator screenOptions={
    {
      header: () => null
    }
  }>
  {
    user?(
            <Stack.Screen name="BottomTabsNavigator" component={
                BottomTabsNavigator } />
          ) : (
  <Stack.Screen
              name= "LoginStack"
component = { LoginStack }
```

```
  />
          )}
</Stack.Navigator>
  < /NavigationContainer>
  < /SafeAreaWrapper>
  );
};

export default Router;



/src/Router / routes.ts


export default {
  MAIN: 'Main',
  TABS: 'Tabs',
  SETTINGS_ROUTER: 'SettingsRouter',
  SETTINGS_SCREEN: 'SettingsScreen',
  LOGIN_STACK: 'SettingsRouter',
  LOGIN_SCREEN: 'SettingsScreen',
  CONFIRM_CODE: 'ConfirmCode',
  MAP: 'Map',
  HISTORY: 'History',
  CONTACTS: 'Contacts',
};


/src/services / authService.ts


import { collections } from "./collections";
import { getAuth } from "./firebaseSetup";

export async function signInWithPhoneNumber(phone: string) {
  try {
    return await getAuth().signInWithPhoneNumber(phone);
  } catch (err) {
    if (
      err.code === "auth/unknown" ||
      err.code === "auth/network-request-failed"
    ) {
      throw new Error("Houve um erro na requisi  o , tente novamente
          mais tarde");
    }
  }
```

```
}



/src/services / firebaseSetup.ts



import "@react-native-firebase/app";
import auth from "@react-native-firebase/auth";
import firestore from "@react-native-firebase/firestore";

const firestoreInstance = firestore();
firestoreInstance.settings({
  persistence: true,
});

export function getAuth() {
  return auth();
}

export function getFirestore() {
  return firestoreInstance;
}

export function getFieldValue() {
  return firestore.FieldValue;
}

export function getCurrentUser() {
  return getAuth().currentUser;
}



/src/services / locationSearch.ts


import axios from 'axios';
import { OPENCAGE_KEY } from 'react-native-dotenv'

async function getLocationName(latitude: number, longitude: number):
    Promise<string> {
  const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude
     }+${longitude}&key=${OPENCAGE_KEY}&language=pt-BR`;

  try {
```

```
    const response = await axios.get(url);

    if (response.data && response.data.results.length > 0) {
      const result = response.data.results[0];
      return result.formatted;
    }

    throw new Error('No results found');
  } catch (error: any) {
    console.error('Error retrieving location name:', error?.message);
    throw error;
  }
}

export {
  getLocationName
}


/src/services / messagingService.ts


import AsyncStorage from "@react-native-async-storage/async-storage";
import messaging from "@react-native-firebase/messaging";
import { FCM_MESSAGES_KEY } from "../utils/constants";
import { getPersistedItem, setPersistentItem } from "../utils/storage";
import { updateUser } from "./userService";

export async function requestUserPermission(): Promise<boolean> {
  try {
    const authStatus = await messaging().requestPermission({
      carPlay: false,
      announcement: true,
    });
    const enabled =
      authStatus === messaging.AuthorizationStatus.AUTHORIZED ||
      authStatus === messaging.AuthorizationStatus.PROVISIONAL;
    if (enabled && !messaging().isDeviceRegisteredForRemoteMessages) {
      messaging().setAutoInitEnabled(true);
    }
    return enabled;
  } catch (error) {
    // Ignore err
  }
  return false;
}

```

```
2529 export async function handleTokenRefresh(): Promise<() => void> {
2530   return messaging().onTokenRefresh((fcmToken) => {
2531     if (fcmToken) {
2532       updateFcmToken(fcmToken);
2533     }
2534   });
2535 }
2536
2537 export async function updateFcmToken(token: string) {
2538   await updateUser({ fcmToken: token });
2539 }
2540
2541 export async function getFcmToken(): Promise<string | null> {
2542   try {
2543     const fcmToken = await messaging().getToken();
2544     if (fcmToken) {
2545       return fcmToken;
2546     }
2547   } catch (error) {
2548     // Ignore err
2549   }
2550   return null;
2551 }
2552
2553 export async function listenToMessages(): Promise<(() => void) |
      undefined> {
2554   try {
2555     const onMessageUnsub = await messaging().onMessage(async ({ data })
         => {
2556       const currentMessages = await getPersistedItem(FCM_MESSAGES_KEY);
2557       if (currentMessages) {
2558         const messageArray = JSON.parse(currentMessages);
2559         messageArray.push(data);
2560         await setPersistentItem(
2561           FCM_MESSAGES_KEY,
2562           JSON.stringify(messageArray),
2563         );
2564       }
2565     });
2566     return onMessageUnsub;
2567   } catch (error) {
2568     // Ignore err
2569   }
2570 }
2571
2572 export async function listenToBackgroundMessages(): Promise<void> {
2573   try {
```

```
    await messaging().setBackgroundMessageHandler(async ({ data }) => {
      const currentMessages = await getPersistedItem(FCM_MESSAGES_KEY);
      if (currentMessages) {
        const messageArray = JSON.parse(currentMessages);
        messageArray.push(data);
        await setPersistentItem(
          FCM_MESSAGES_KEY,
          JSON.stringify(messageArray),
        );
      }
    });
  } catch (error) {
    // Ignore err
  }
}

export function sendSMS(contacts: string[], message: string) {
  Linking.openURL(`sms:${contacts}?body=${message}`)
}


/ src / services / locationService

import axios from 'axios';
import { OPENCAGE_KEY } from '@env'

export async function getLocationDetails(latitude: number, longitude:
    number): Promise<string> {
  const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude
      }+${longitude}&key=${OPENCAGE_KEY}&language=pt-BR`
  console.log(`URL::`, url)

  try {
    const response = await axios.get(url);

    if (response.data && response.data.results.length > 0) {
      const result = response.data.results[0];
      return result.formatted;
    }

    throw new Error('No results found');
  } catch (error: any) {
    console.error('Error retrieving location name:', error.message);
    throw error;
  }
}

```

```
2619
2620 export async function getCityName(latitude: number, longitude: number):
        Promise<{ city: string, state: string }> {
2621   const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude
         }+${longitude}&key=${OPENCAGE_KEY}&language=pt-BR`
2622   console.log(`URL::`, url)
2623
2624   try {
2625     const response = await axios.get(url);
2626
2627     if (response.data && response.data.results.length > 0) {
2628       const result = response.data.results[0];
2629       const components = result.components;
2630       const city = components.city || components.town || components.
             village;
2631       const state = components.state;
2632
2633       if (city && state) {
2634         return { city, state };
2635       }
2636     }
2637
2638     throw new Error('No results found');
2639   } catch (error: any) {
2640     console.error('Error retrieving location name:', error.message);
2641     throw error;
2642   }
2643 }
2644
2645
2646 / src / services / openMeteo.ts
2647
2648
2649 import axios from 'axios';
2650
2651 export async function fetchWeatherData(city) {
2652   const apiUrl = `https://api.open-meteo.com/weather?location=${city}`;
2653
2654   try {
2655     const response = await axios.get(apiUrl);
2656     return response.data;
2657   } catch (error) {
2658     console.error('Error fetching weather data:', error);
2659     throw error;
2660   }
2661 }
2662
```

```
2663 export function checkForWeatherDangers(weatherData) {
2664   const weatherConditions = weatherData.weather.code;
2665   const warnings = [];
2666
2667   if (weatherConditions === 5 || weatherConditions === 6) {
2668     warnings.push('Perigo! Tempestade a caminho!');
2669   }
2670
2671   if (weatherConditions === 9 || weatherConditions === 10) {
2672     warnings.push('Perigo! Pode haver um tornado pr ximo a voc !');
2673   }
2674
2675   if (weatherConditions === 16) {
2676     warnings.push('Perigo! Inunda  o em sua regi o!');
2677   }
2678
2679   if (weatherConditions === 25) {
2680     warnings.push('Perigo! Chuva forte a caminho!');
2681   }
2682
2683   if (weatherConditions === 26) {
2684     warnings.push('Perigo! Uma nevasca se aproxima!');
2685   }
2686
2687   return warnings;
2688 }
2689
2690
2691
2692 /src/services / userService.ts
2693
2694
2695 import { FirebaseFirestoreTypes } from "@react-native-firebase/firestore
       ";
2696 import { getPhoneNumber } from "../utils/deviceHelper";
2697 import { collections, subCollections } from "./collections";
2698
2699 export async function updateUser(data: any) {
2700   const phoneNumber = await getPhoneNumber();
2701   await collections.users.doc(phoneNumber).update(data);
2702 }
2703
2704 export async function getFavoriteContacts() {
2705   const accounts: Array<any> = [];
2706   const phoneNumber = await getPhoneNumber();
2707   const subDoc = await subCollections.favoriteContacts(phoneNumber).get
       ();
```

```
2708   subDoc.forEach((doc) => accounts.push(doc.data()));
2709   return accounts;
2710 }
2711
2712 export async function getUser() {
2713   const phoneNumber = await getPhoneNumber();
2714   const result = await collections.users.doc(phoneNumber).get();
2715   return result.data();
2716 }
2717
2718 export async function updateAccountFromUser({ accountName, data }: {
2719   accountName: string,
2720   data: Partial<{
2721     [x: string]: any;
2722   }>
2723 }) {
2724   const phoneNumber = await getPhoneNumber();
2725   const subDoc = await subCollections
2726     .favoriteContacts(phoneNumber)
2727     .doc(accountName)
2728     .get();
2729   if (subDoc.exists) {
2730     return await subCollections
2731       .favoriteContacts(phoneNumber)
2732       .doc(accountName)
2733       .update({ accountName, ...data });
2734   } else {
2735     return await subCollections
2736       .favoriteContacts(phoneNumber)
2737       .doc(accountName)
2738       .set({ accountName, ...data });
2739   }
2740 }
2741
2742 export async function removeAccountFromUser(accountName: string) {
2743   const phoneNumber = await getPhoneNumber();
2744   await subCollections.favoriteContacts(phoneNumber).doc(accountName).
       delete();
2745 }
2746
2747 export async function subscribeToUser(handler: (d?:
      FirebaseFirestoreTypes.DocumentData) => void, handleError = (err) =>
      null) {
2748   const phoneNumber = await getPhoneNumber();
2749   collections.users.doc(phoneNumber).onSnapshot(
2750     { includeMetadataChanges: false },
2751     (doc) => {
```

```
2752        if (doc.exists) {
2753          handler(doc.data());
2754        }
2755      },
2756      (err) => {
2757        handleError(err);
2758      },
2759    );
2760  }
2761
2762  export async function subscribeToFavoriteContacts(handler: (accs:
       FirebaseFirestoreTypes.DocumentData[]) => void, handleError = (err)
       => null) {
2763    const phoneNumber = await getPhoneNumber();
2764    return subCollections.favoriteContacts(phoneNumber).onSnapshot(
2765      { includeMetadataChanges: false },
2766      (docs) => {
2767        const accounts: Array<FirebaseFirestoreTypes.DocumentData> = [];
2768        docs.forEach((doc) => accounts.push(doc.data()));
2769        handler(accounts);
2770      },
2771      (err) => {
2772        handleError(err);
2773      },
2774    );
2775  }
2776
2777  export const updateFirebaseDatabase = async (phoneNumber, fcmToken) => {
2778    const databaseRef = firebase.database().ref('users');
2779
2780    const userId = firebase.auth().currentUser.uid;
2781    await databaseRef.child(userId).set({
2782      phoneNumber: phoneNumber,
2783      fcmToken: fcmToken,
2784    })
2785  }
2786
2787
2788
2789  functions / src / firebaseAdmin.ts
2790
2791
2792
2793  const functions = require('firebase-functions');
2794  const admin = require('firebase-admin');
2795
2796  admin.initializeApp();
```

```
2797
2798 const sendPushNotifications =
2799   functions.https.onRequest(async (req, res) => {
2800     const phoneNumbers = req.body.phoneNumbers;
2801     const payload = {
2802       notification: {
2803         title: 'Alerta!',
2804         body: `${req.body.name} lhe informa: ${req.body.message}`,
2805       },
2806     };
2807
2808     try {
2809       const tokensSnapshot = await admin
2810         .database()
2811         .ref('users')
2812         .once('value');
2813       const tokens = [];
2814
2815       tokensSnapshot.forEach((childSnapshot) => {
2816         const user = childSnapshot.val();
2817         if (phoneNumbers.includes(user.phoneNumber) && user.fcmToken) {
2818           tokens.push(user.fcmToken);
2819         }
2820       });
2821
2822       if (tokens.length > 0) {
2823         await admin.messaging().sendToDevice(tokens, payload);
2824         res.status(200).send('Notifica  es enviadas com sucesso');
2825       } else {
2826         const phones = phoneNumbers.join(',')
2827         res.status(200)
2828           .send(
2829             `Estes contatos n o possuem o app instalado: ${phones}`
2830           );
2831       }
2832     } catch (error) {
2833       console.error('Error sending push notifications:', error);
2834       res.status(500).send('Error sending push notifications.');
2835     }
2836   });
2837
2838
2839
2840 /src/theme / styles.ts
2841
2842
2843 export const commonColors = {
```

```
2844    transparent: 'transparent',
2845    red: 'red',
2846    black: '#000000',
2847    white: '#fff',
2848    primary: '#702EDA',
2849    color1: '#00DA55',
2850    color2: '#3BCC73',
2851    color3: '#AAAAAA',
2852    color4: '#27AE60',
2853    color5: '#828282',
2854    color6: 'rgba(255, 255, 255, 0.85)',
2855    color7: '#ededed',
2856    color8: '#A25DEB',
2857    color9: '#6F2DD9',
2858    color10: 'rgba(255, 255, 255, 0.6)',
2859    color11: '#767676',
2860    color12: '#BDBDBD',
2861    color13: '#54C96D',
2862    color14: '#1155E7',
2863    color15: '#C4C4C4',
2864    color16: 'rgba(0, 0, 0, 0.1)',
2865    color17: '#F2F2F2',
2866    color18: '#AAAAAA',
2867    color19: '#8F58E7',
2868    color20: '#4b4c4e',
2869    color21: '#898A8D',
2870    color22: 'rgba(112, 46, 218, 0.36)',
2871    color23: '#6639D1',
2872    color24: 'rgba(0, 0, 0, 0.75)',
2873    color25: '#FFD700'
2874  };
2875
2876  const lightTheme = {
2877    ...commonColors,
2878    bgColor: commonColors.white,
2879    screenBgColor: commonColors.white,
2880    text: commonColors.color11,
2881    selected: commonColors.color9,
2882  };
2883
2884  const darkTheme = {
2885    ...commonColors,
2886    bgColor: commonColors.primary,
2887    screenBgColor: commonColors.color24,
2888    text: commonColors.white,
2889    selected: commonColors.white,
2890  };
```

```
2891
2892 const theme = {
2893   colors: {
2894     dark: darkTheme,
2895     light: lightTheme,
2896   },
2897 };
2898
2899 export default theme;
2900
2901
2902
2903 /src/types / context.d.ts
2904
2905
2906 import { FirebaseAuthTypes } from "@react-native-firebase/auth";
2907
2908 export type ThemeProps = 'dark' | 'light';
2909
2910 export type UserContextType = {
2911   user?: FirebaseAuthTypes.User | null;
2912   hasVerification: boolean;
2913   signIn: (number: string) => void;
2914   signOut: () => void;
2915   confirmCode: (code: string) => void;
2916 }
2917
2918
2919 /src/types / styles.d.ts
2920
2921
2922 export type ThemeObject = {
2923   [key: string]: string;
2924 };
2925
2926
2927
2928 /src/utils / constants.ts
2929
2930
2931 import { Dimensions, PixelRatio, Platform } from 'react-native';
2932 import { Contact } from 'react-native-contacts';
2933 import { remove } from 'remove-accents';
2934
2935 const screen = Dimensions.get('screen');
2936 export const screenWidth = screen['width'];
2937 export const screenHeight = screen['height'];
```

```
2938
2939 export const scaleToScreen = (value: number): number =>
2940   PixelRatio.roundToNearestPixel((value * screenWidth) / 414);
2941
2942 export const REPORT_BUTTON_SIZE = 50;
2943 export const REPORT_BUTTON_POSITION = (screenWidth / 2) - (
       REPORT_BUTTON_SIZE / 2);
2944
2945 export const isIOS = Platform.OS === 'ios';
2946
2947 export const NOTCH_DEFAULT_PADDING = 30;
2948
2949 export const SAFE_AREA_IOS_PADDING = 20;
2950
2951 export const SAFE_AREA_ANDROID_PADDING = 12;
2952
2953 export const BOTTOM_TAB_HEIGHT = isIOS ? 85 : 65;
2954
2955 export const COMMON_ICON_SIZE = 26;
2956
2957 export const THEMES = {
2958   LIGHT: 'light',
2959   DARK: 'dark',
2960 };
2961
2962 export const INITIAL_POSITION = {
2963   lat: -23.556831106,
2964   lng: -46.653830718
2965 }
2966
2967 export const TAB_NAMES = {
2968   MAP: 'Mapa',
2969   SETTINGS: 'Configura  es'
2970 }
2971
2972 export const FCM_MESSAGES_KEY = "FCM_MESSAGES_KEY";
2973
2974 export const getContactName = (c: Contact) => {
2975   return remove(`${c.givenName} ${c.familyName}`)
2976 }
2977
2978 export const PHONE_AUTH_CONSTANT = `STORAGE/PHONE_AUTH_CONSTANT`
2979
2980 export const USER_UUID_STORAGE_KEY = `USER_UUID_STORAGE_KEY`
2981
2982 export const BRAZILIAN_PHONE_CODE = `+55`
2983
```

```
2984
2985   / src / utils / deviceHelper.ts
2986
2987
2988 import { hasNotch } from 'react-native-device-info';
2989 import {
2990   isIOS,
2991   NOTCH_DEFAULT_PADDING,
2992   PHONE_AUTH_CONSTANT,
2993   SAFE_AREA_ANDROID_PADDING,
2994   SAFE_AREA_IOS_PADDING,
2995 } from './constants';
2996 import { getPersistedItem } from './storage';
2997
2998 export function getNotchPadding(topInset: number): number {
2999   if (topInset > 24 || hasNotch()) {
3000     return NOTCH_DEFAULT_PADDING;
3001   } else {
3002     return isIOS ? SAFE_AREA_IOS_PADDING : SAFE_AREA_ANDROID_PADDING;
3003   }
3004 }
3005
3006 export async function getPhoneNumber() {
3007   return await getPersistedItem(PHONE_AUTH_CONSTANT) ?? ''
3008 }
3009
3010 export const isBrazilianPhone = (phone: string): boolean => {
3011   const phoneRegex = /^\+55\s?\(?([1-9]{2})\)?\s?([9]?\d{4})-?(\d{4})$/;
3012   return phoneRegex.test(phone);
3013 };
3014
3015 export const formatPhoneNumber = (input: string): string => {
3016   const digitsOnly = input.replace(/\D/g, '');
3017
3018   if (digitsOnly.length === 0) {
3019     return '';
3020   }
3021
3022   let formattedNumber = '';
3023   if (digitsOnly.length <= 2) {
3024     formattedNumber = digitsOnly;
3025   } else if (digitsOnly.length <= 6) {
3026     formattedNumber = `(${digitsOnly.slice(0, 2)}) ${digitsOnly.slice(2)
          }`;
3027   } else if (digitsOnly.length <= 10) {
3028     formattedNumber = `(${digitsOnly.slice(0, 2)}) ${digitsOnly.slice(2,
          6)}-${digitsOnly.slice(6)}`;
```

```
  } else {
    formattedNumber = `(${digitsOnly.slice(0, 2)}) ${digitsOnly.slice(2,
        7)}-${digitsOnly.slice(7, 11)}`;
  }

  return formattedNumber;
};


/src/utils / enums.ts


export enum PERMISSIONS {
  NOTIFICATIONS = 'NOTIFICATIONS',
  LOCATION_ACC = 'LOCATION_ACC',
  CONTACTS = 'CONTACTS'
}

export enum PermissionsResponse {
  GRANTED = "granted",
  LIMITED = "limited",
  UNAVAILABLE = "unavailable",
  BLOCKED = "blocked",
  DENIED = "denied"
}


/src/utils / storage.ts


import AsyncStorage from "@react-native-async-storage/async-storage";

export const setPersistentItem = async (key: string, value: string) => {
  try {
    await AsyncStorage.setItem(key, value)
  } catch (error) {

  }
}

export const getPersistedItem = async (key: string) => {
  try {
    return await AsyncStorage.getItem(key)
  } catch (error) {

  }
}
```

```
3075
3076 export const deletePersistedItem = async (key: string) => {
3077   try {
3078     return await AsyncStorage.removeItem(key)
3079   } catch (error) {
3080
3081   }
3082 }
3083
3084
3085 /src/utils / routesHelper.ts
3086
3087
3088 import { BackHandler } from 'react-native';
3089
3090 export function disableHardwareBackButton() {
3091   BackHandler.addEventListener('hardwareBackPress', () => true);
3092 }
3093
3094 export function removeDisableHardwareBackButtonListener() {
3095   BackHandler.removeEventListener('hardwareBackPress', () => true);
3096 }
3097
3098
3099
3100 /src/App.tsx
3101
3102
3103 import React from 'react';
3104
3105 import { ContactsProvider } from './contexts/ContactsContext';
3106 import PermissionsProvider from './contexts/PermissionsContext';
3107 import ThemeProvider from './contexts/ThemeContext';
3108 import UserProvider from './contexts/UserContext';
3109
3110 import useAlertsSubscription from './hooks/useAlertsSubscription';
3111 import usePermissions from './hooks/usePermissions';
3112 import useCloudMessaging from './hooks/useCloudMessaging';
3113
3114 import Routes from './Router/Router';
3115 import './services/firebaseSetup';
3116 import { SafeAreaProvider } from 'react-native-safe-area-context';
3117 import { isIOS } from './utils/constants';
3118 import StatusBar from './components/StatusBar/CustomStatusBar';
3119
3120 const App = () => {
3121   usePermissions();
```

```
  useCloudMessaging();
  useAlertsSubscription();

  return (
    <SafeAreaProvider>
    <PermissionsProvider>
    <ThemeProvider>
    <UserProvider>
    <ContactsProvider>
    <StatusBar
                hidden= { false}
  animated = { true}
  barStyle = { 'light-content'}
  backgroundColor = {`#FFF`
}
/>
  < Routes />
  </ContactsProvider>
  < /UserProvider>
  < /ThemeProvider>
  < /PermissionsProvider>
  < /SafeAreaProvider>
  );
};

export default App;



/alerta-civil-firebase/src / index.ts


import { onRequest } from "firebase-functions/v2/https";
import { database, messaging } from "firebase-admin";

// Start writing functions
// https://firebase.google.com/docs/functions/typescript

export const sendPushNotifications = onRequest(async (req, res) => {
  const phoneNumbers = req.body.phoneNumbers;
  const payload = {
    notification: {
      title: "Alerta!",
      body: `${req.body.name} lhe informa: ${req.body.message}`,
    },
  };

```

```
3169   try {
3170     const tokensSnapshot = await database().ref("users").once("value");
3171     const tokens: string[] = [];
3172
3173     tokensSnapshot.forEach((childSnapshot: any) => {
3174       const user = childSnapshot.val();
3175       if (phoneNumbers.includes(user.phoneNumber) && user.fcmToken) {
3176         tokens.push(user.fcmToken);
3177       }
3178     });
3179     const err = "No matching FCM tokens found for the given phone
          numbers.";
3180     if (tokens.length > 0) {
3181       await messaging().sendToDevice(tokens, payload);
3182       res.status(200).send("Push notification sent successfully!");
3183     } else {
3184       res.status(200).send(err);
3185     }
3186   } catch (error) {
3187     console.error("Error sending push notifications:", error);
3188     res.status(500).send("Error sending push notifications.");
3189   }
3190 });
3191
3192
3193
3194 /alerta-civil-firebase/.gitignore
3195
3196
3197 # Compiled JavaScript files
3198 lib/**/ *.js
3199 lib/**/ *.js.map
3200
3201 # TypeScript v1 declaration files
3202 typings /
3203
3204 # Node.js dependency directory
3205 node_modules /
3206
3207
3208
3209   /alerta-civil-firebase.eslintrc.js
3210
3211
3212 module.exports = {
3213   root: true,
3214   env: {
```

```
    es6: true,
    node: true,
  },
  extends: [
    "eslint:recommended",
    "plugin:import/errors",
    "plugin:import/warnings",
    "plugin:import/typescript",
    "google",
    "plugin:@typescript-eslint/recommended",
  ],
  parser: "@typescript-eslint/parser",
  parserOptions: {
    project: [
      "tsconfig.json",
      "tsconfig.dev.json",
    ],
    sourceType: "module",
    tsconfigRootDir: __dirname,
  },
  ignorePatterns: [
    "/lib/**/*", // Ignore built files.
  ],
  plugins: [
    "@typescript-eslint",
    "import",
  ],
  rules: {
    semi: "off",
    indent: "off",
    "no-tabs": "off",
    "no-mixed-spaces-and-tabs": "off",
    "object-curly-spacing": "off",
    "require-jsdoc": "off",
    "@typescript-eslint/explicit-module-boundary-types": "off",
    "operator-linebreak": "off",
    quotes: "off",
    "@typescript-eslint/no-explicit-any": "off",
    "quote-props": "off",
  },
};



/alerta-civil-firebase/tsconfig.json

```

```
{
  "compilerOptions": {
    "module": "commonjs",
      "noImplicitReturns": false,
        "noUnusedLocals": true,
          "outDir": "lib",
            "sourceMap": true,
              "strict": true,
                "target": "es2020",
                  "esModuleInterop": true,
                    "noImplicitAny": false,
                      "skipLibCheck": true,
                        "rootDir": "src"
  },
  "compileOnSave": true,
    "include": [
      "src"
    ],
      "exclude": [
        "node_modules",
        "lib"
      ]
}


/alerta-civil-firebase/package.json


{
  "name": "functions",
    "scripts": {
    "lint": "eslint --ext .js,.ts .",
      "build": "tsc",
        "build:watch": "tsc --watch",
          "serve": "npm run build && firebase emulators:start --only
             functions",
            "shell": "npm run build && firebase functions:shell",
              "start": "npm run shell",
                "deploy": "firebase deploy --only functions",
                  "logs": "firebase functions:log"
  },
  "engines": {
    "node": "18"
  },
  "main": "lib/index.js",
    "dependencies": {
    "firebase-admin": "^11.8.0",
```

```
3308       "firebase-functions": "^4.3.1"
3309   },
3310   "devDependencies": {
3311     "@typescript-eslint/eslint-plugin": "^5.12.0",
3312       "@typescript-eslint/parser": "^5.12.0",
3313         "eslint": "^8.9.0",
3314           "eslint-config-google": "^0.14.0",
3315             "eslint-plugin-import": "^2.25.4",
3316               "firebase-functions-test": "^3.1.0",
3317                 "typescript": "^4.9.0"
3318   },
3319   "private": true
3320 }
3321
3322
3323
3324 \textbf{ /android/build.gradle }
3325
3326
3327 // Top-level build file where you can add configuration options common
       to all sub-projects/modules.
3328
3329 buildscript {
3330     ext {
3331     buildToolsVersion = "31.0.0"
3332     minSdkVersion = 21
3333     compileSdkVersion = 31
3334     targetSdkVersion = 31
3335     googlePlayServicesVersion = "17.0.0"
3336     androidMapsUtilsVersion = "0.5+"
3337
3338     if (System.properties['os.arch'] == "aarch64") {
3339       // For M1 Users we need to use the NDK 24 which added support for
             aarch64
3340       ndkVersion = "24.0.8215888"
3341     } else {
3342       // Otherwise we default to the side-by-side NDK version from AGP.
3343       ndkVersion = "21.4.7075529"
3344     }
3345   }
3346     repositories {
3347     google()
3348     mavenCentral()
3349   }
3350     dependencies {
3351     classpath("com.android.tools.build:gradle:7.1.1")
3352     classpath("de.undercouch:gradle-download-task:5.0.1")
```

```
3353     classpath("com.google.gms:google-services:4.3.14")
3354     // NOTE: Do not place your application dependencies here; they
             belong
3355     // in the individual module build.gradle files
3356   }
3357 }
3358
3359 allprojects {
3360     repositories {
3361         exclusiveContent {
3362             filter {
3363                 includeGroup "com.facebook.react"
3364       }
3365             forRepository {
3366                 maven {
3367                     url "$rootDir/../node_modules/react-native/android"
3368         }
3369       }
3370     }
3371         maven {
3372       // All of React Native (JS, Obj-C sources, Android binaries) is
             installed from npm
3373       url("$rootDir/../node_modules/react-native/android")
3374     }
3375         maven {
3376       // Android JSC is installed from npm
3377       url("$rootDir/../node_modules/jsc-android/dist")
3378     }
3379         mavenCentral {
3380             // We don't want to fetch react-native from Maven Central as
                 there are
3381             // older versions over there.
3382             content {
3383                 excludeGroup "com.facebook.react"
3384       }
3385     }
3386     mavenCentral()
3387     google()
3388         maven { url 'https://www.jitpack.io' }
3389   }
3390 }
3391
3392
3393
3394 \textbf{ /android/app / build.gradle }
3395
3396
```

```
3397     apply plugin: 'com.android.application'
3398 apply plugin: 'com.google.gms.google-services'
3399
3400 import com.android.build.OutputFile
3401 import org.apache.tools.ant.taskdefs.condition.Os
3402
3403 /**
3404  * The react.gradle file registers a task for each build variant (e.g.
        bundleDebugJsAndAssets
3405  * and bundleReleaseJsAndAssets).
3406  * These basically call `react-native bundle` with the correct arguments
        during the Android build
3407  * cycle. By default, bundleDebugJsAndAssets is skipped, as in debug/dev
        mode we prefer to load the
3408  * bundle directly from the development server. Below you can see all
        the possible configurations
3409  * and their defaults. If you decide to add a configuration block, make
        sure to add it before the
3410  * `apply from: "../../node_modules/react-native/react.gradle"` line.
3411  *
3412  * project.ext.react = [
3413  *   // the name of the generated asset file containing your JS bundle
3414  *   bundleAssetName: "index.android.bundle",
3415  *
3416  *   // the entry file for bundle generation. If none specified and
3417  *   // "index.android.js" exists, it will be used. Otherwise "index.js"
        is
3418  *   // default. Can be overridden with ENTRY_FILE environment variable.
3419  *   entryFile: "index.android.js",
3420  *
3421  *   // https://reactnative.dev/docs/performance#enable-the-ram-format
3422  *   bundleCommand: "ram-bundle",
3423  *
3424  *   // whether to bundle JS and assets in debug mode
3425  *   bundleInDebug: false,
3426  *
3427  *   // whether to bundle JS and assets in release mode
3428  *   bundleInRelease: true,
3429  *
3430  *   // whether to bundle JS and assets in another build variant (if
        configured).
3431  *   // See http://tools.android.com/tech-docs/new-build-system/user-
        guide#TOC-Build-Variants
3432  *   // The configuration property can be in the following formats
3433  *   //         'bundleIn${productFlavor}${buildType}'
3434  *   //         'bundleIn${buildType}'
3435  *   // bundleInFreeDebug: true,
```

```
3436 *   // bundleInPaidRelease: true,
3437 *   // bundleInBeta: true,
3438 *
3439 *   // whether to disable dev mode in custom build variants (by default
     only disabled in release)
3440 *   // for example: to disable dev mode in the staging build type (if
     configured)
3441 *   devDisabledInStaging: true,
3442 *   // The configuration property can be in the following formats
3443 *   //         'devDisabledIn${productFlavor}${buildType}'
3444 *   //         'devDisabledIn${buildType}'
3445 *
3446 *   // the root of your project, i.e. where "package.json" lives
3447 *   root: "../../",
3448 *
3449 *   // where to put the JS bundle asset in debug mode
3450 *   jsBundleDirDebug: "$buildDir/intermediates/assets/debug",
3451 *
3452 *   // where to put the JS bundle asset in release mode
3453 *   jsBundleDirRelease: "$buildDir/intermediates/assets/release",
3454 *
3455 *   // where to put drawable resources / React Native assets, e.g. the
     ones you use via
3456 *   // require('./image.png')), in debug mode
3457 *   resourcesDirDebug: "$buildDir/intermediates/res/merged/debug",
3458 *
3459 *   // where to put drawable resources / React Native assets, e.g. the
     ones you use via
3460 *   // require('./image.png')), in release mode
3461 *   resourcesDirRelease: "$buildDir/intermediates/res/merged/release",
3462 *
3463 *   // by default the gradle tasks are skipped if none of the JS files
     or assets change; this means
3464 *   // that we don't look at files in android/ or ios/ to determine
     whether the tasks are up to
3465 *   // date; if you have any other folders that you want to ignore for
     performance reasons (gradle
3466 *   // indexes the entire tree), add them here. Alternatively, if you
     have JS files in android/
3467 *   // for example, you might want to remove it from here.
3468 *   inputExcludes: ["android/**", "ios/**"],
3469 *
3470 *   // override which node gets called and with what additional
     arguments
3471 *   nodeExecutableAndArgs: ["node"],
3472 *
3473 *   // supply additional arguments to the packager
```

```
3474  *   extraPackagerArgs: []
3475  * ]
3476  */
3477
3478 project.ext.react = [
3479   enableHermes: false,  // clean and rebuild if changing
3480 ]
3481
3482 apply from: "../../node_modules/react-native/react.gradle"
3483
3484 /**
3485  * Set this to true to create two separate APKs instead of one:
3486  *   - An APK that only works on ARM devices
3487  *   - An APK that only works on x86 devices
3488  * The advantage is the size of the APK is reduced by about 4MB.
3489  * Upload all the APKs to the Play Store and people will download
3490  * the correct one based on the CPU architecture of their device.
3491  */
3492 def enableSeparateBuildPerCPUArchitecture = false
3493
3494 /**
3495  * Run Proguard to shrink the Java bytecode in release builds.
3496  */
3497 def enableProguardInReleaseBuilds = false
3498
3499 /**
3500  * The preferred build flavor of JavaScriptCore.
3501  *
3502  * For example, to use the international variant, you can use:
3503  * `def jscFlavor = 'org.webkit:android-jsc-intl:+'`
3504  *
3505  * The international variant includes ICU i18n library and necessary
        data
3506  * allowing to use e.g. `Date.toLocaleString` and `String.localeCompare`
        that
3507  * give correct results when using with locales other than en-US.  Note
        that
3508  * this variant is about 6MiB larger per architecture than default.
3509  */
3510 def jscFlavor = 'org.webkit:android-jsc:+'
3511
3512 /**
3513  * Whether to enable the Hermes VM.
3514  *
3515  * This should be set on project.ext.react and that value will be read
        here. If it is not set
3516  * on project.ext.react, JavaScript will not be compiled to Hermes
```

```
    Bytecode
 * and the benefits of using Hermes will therefore be sharply reduced.
 */
def enableHermes = project.ext.react.get("enableHermes", false);

/**
 * Architectures to build native code for.
 */
def reactNativeArchitectures() {
    def value = project.getProperties().get("reactNativeArchitectures")
  return value ? value.split(",") : ["armeabi-v7a", "x86", "x86_64", "
      arm64-v8a"]
}

def isNewArchitectureEnabled() {
  // To opt-in for the New Architecture, you can either:
  // - Set `newArchEnabled` to true inside the `gradle.properties` file
  // - Invoke gradle with `-newArchEnabled=true`
  // - Set an environment variable `ORG_GRADLE_PROJECT_newArchEnabled=
      true`
  return project.hasProperty("newArchEnabled") && project.newArchEnabled
      == "true"
}

android {
    ndkVersion rootProject.ext.ndkVersion

    compileSdkVersion rootProject.ext.compileSdkVersion

    defaultConfig {
        applicationId "com.alertacivil"
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion
        versionCode 1
        versionName "1.0"
        buildConfigField "boolean", "IS_NEW_ARCHITECTURE_ENABLED",
            isNewArchitectureEnabled().toString()

    if (isNewArchitectureEnabled()) {
            // We configure the NDK build only if you decide to opt-in
                for the New Architecture.
            externalNativeBuild {
                cmake {
                    arguments "-DPROJECT_BUILD_DIR=$buildDir",
            "-DREACT_ANDROID_DIR=$rootDir/../node_modules/react-native/
                ReactAndroid",
            "-DREACT_ANDROID_BUILD_DIR=$rootDir/../node_modules/react-
```

```
                native/ReactAndroid/build",
3557            "-DNODE_MODULES_DIR=$rootDir/../node_modules",
3558            "-DANDROID_STL=c++_shared"
3559        }
3560      }
3561      if (!enableSeparateBuildPerCPUArchitecture) {
3562              ndk {
3563          abiFilters(* reactNativeArchitectures())
3564        }
3565      }
3566    }
3567  }
3568
3569  if (isNewArchitectureEnabled()) {
3570        // We configure the NDK build only if you decide to opt-in for
              the New Architecture.
3571        externalNativeBuild {
3572            ndkBuild {
3573                path "$projectDir/src/main/jni/CMakeLists.txt"
3574      }
3575    }
3576        def reactAndroidProjectDir = project(':ReactAndroid').projectDir
3577        def packageReactNdkDebugLibs = tasks.register("
              packageReactNdkDebugLibs", Copy) {
3578      dependsOn(":ReactAndroid:packageReactNdkDebugLibsForBuck")
3579      from("$reactAndroidProjectDir/src/main/jni/prebuilt/lib")
3580      into("$buildDir/react-ndk/exported")
3581    }
3582        def packageReactNdkReleaseLibs = tasks.register("
              packageReactNdkReleaseLibs", Copy) {
3583      dependsOn(":ReactAndroid:packageReactNdkReleaseLibsForBuck")
3584      from("$reactAndroidProjectDir/src/main/jni/prebuilt/lib")
3585      into("$buildDir/react-ndk/exported")
3586    }
3587        afterEvaluate {
3588      // If you wish to add a custom TurboModule or component locally,
3589      // you should uncomment this line.
3590      // preBuild.dependsOn("generateCodegenArtifactsFromSchema")
3591      preDebugBuild.dependsOn(packageReactNdkDebugLibs)
3592      preReleaseBuild.dependsOn(packageReactNdkReleaseLibs)
3593
3594      // Due to a bug inside AGP, we have to explicitly set a dependency
3595      // between configureNdkBuild* tasks and the preBuild tasks.
3596      // This can be removed once this is solved: https://issuetracker.
              google.com/issues/207403732
3597      configureNdkBuildRelease.dependsOn(preReleaseBuild)
3598      configureNdkBuildDebug.dependsOn(preDebugBuild)
```

```
3599      reactNativeArchitectures().each {
3600        architecture ->
3601          tasks.findByName("configureNdkBuildDebug[${architecture}]")?.
                configure {
3602          dependsOn("preDebugBuild")
3603        }
3604        tasks.findByName("configureNdkBuildRelease[${architecture}]")?.
               configure {
3605          dependsOn("preReleaseBuild")
3606        }
3607      }
3608    }
3609  }
3610
3611    splits {
3612        abi {
3613      reset()
3614            enable enableSeparateBuildPerCPUArchitecture
3615            universalApk false  // If true, also generate a universal
                APK
3616      include(* reactNativeArchitectures())
3617    }
3618  }
3619    signingConfigs {
3620        debug {
3621            storeFile file('debug.keystore')
3622            storePassword 'android'
3623            keyAlias 'androiddebugkey'
3624            keyPassword 'android'
3625    }
3626  }
3627    buildTypes {
3628        debug {
3629            signingConfig signingConfigs.debug
3630    }
3631        release {
3632            // Caution! In production, you need to generate your own
                keystore file.
3633            // see https://reactnative.dev/docs/signed-apk-android.
3634            signingConfig signingConfigs.debug
3635            minifyEnabled enableProguardInReleaseBuilds
3636            proguardFiles getDefaultProguardFile("proguard-android.txt")
                , "proguard-rules.pro"
3637    }
3638  }
3639
3640  // applicationVariants are e.g. debug, release
```

```
3641   applicationVariants.all {
3642     variant ->
3643       variant.outputs.each {
3644       output ->
3645         // For each separate APK per architecture, set a unique version
               code as described here:
3646         // https://developer.android.com/studio/build/configure-apk-
               splits.html
3647         // Example: versionCode 1 will generate 1001 for armeabi-v7a,
               1002 for x86, etc.
3648         def versionCodes = ["armeabi-v7a": 1, "x86": 2, "arm64-v8a": 3,
               "x86_64": 4]
3649             def abi = output.getFilter(OutputFile.ABI)
3650       if (abi != null) {  // null for the universal-debug, universal-
             release variants
3651         output.versionCodeOverride =
3652           defaultConfig.versionCode * 1000 + versionCodes.get(abi)
3653       }
3654
3655     }
3656   }
3657 }
3658
3659 dependencies {
3660     implementation fileTree(dir: "libs", include: ["*.jar"])
3661
3662     //noinspection GradleDynamicVersion
3663     implementation "com.facebook.react:react-native:+"  // From
           node_modules
3664
3665     implementation "androidx.swiperefreshlayout:swiperefreshlayout:1.0.0
           "
3666     implementation platform('com.google.firebase:firebase-bom:32.1.0')
3667     implementation 'com.google.firebase:firebase-auth'
3668     implementation project(':react-native-contacts')
3669     implementation project(':react-native-maps')
3670
3671   debugImplementation("com.facebook.flipper:flipper:${FLIPPER_VERSION}")
         {
3672         exclude group: 'com.facebook.fbjni'
3673   }
3674
3675   debugImplementation("com.facebook.flipper:flipper-network-plugin:${
       FLIPPER_VERSION}") {
3676         exclude group: 'com.facebook.flipper'
3677         exclude group: 'com.squareup.okhttp3', module: 'okhttp'
3678   }
```

```
3679
3680   debugImplementation("com.facebook.flipper:flipper-fresco-plugin:${
           FLIPPER_VERSION}") {
3681       exclude group: 'com.facebook.flipper'
3682   }
3683
3684   if (enableHermes) {
3685     //noinspection GradleDynamicVersion
3686     implementation("com.facebook.react:hermes-engine:+") { // From
           node_modules
3687         exclude group: 'com.facebook.fbjni'
3688     }
3689   } else {
3690       implementation jscFlavor
3691   }
3692 }
3693
3694 if (isNewArchitectureEnabled()) {
3695   // If new architecture is enabled, we let you build RN from source
3696   // Otherwise we fallback to a prebuilt .aar bundled in the NPM package
         .
3697   // This will be applied to all the imported transtitive dependency.
3698   configurations.all {
3699     resolutionStrategy.dependencySubstitution {
3700       substitute(module("com.facebook.react:react-native"))
3701         .using(project(":ReactAndroid"))
3702         .because("On New Architecture we're building React Native from
              source")
3703       substitute(module("com.facebook.react:hermes-engine"))
3704         .using(project(":ReactAndroid:hermes-engine"))
3705         .because("On New Architecture we're building Hermes from source"
              )
3706     }
3707   }
3708 }
3709
3710 // Run this once to be able to run the application with BUCK
3711 // puts all compile dependencies into folder libs for BUCK to use
3712 task copyDownloadableDepsToLibs(type: Copy) {
3713     from configurations.implementation
3714     into 'libs'
3715 }
3716
3717 apply from: file("../../node_modules/@react-native-community/cli-
       platform-android/native_modules.gradle");
     applyNativeModulesAppBuildGradle(project)
3718
```

```
3719
3720
3721
3722   / ios / Podfile
3723
3724
3725     require File.join(File.dirname(`node --print "require.resolve('expo/
           package.json')"`), "scripts/autolinking")
3726 require_relative '../node_modules/react-native/scripts/react_native_pods
       '
3727 require_relative '../node_modules/@react-native-community/cli-platform-
       ios/native_modules'
3728
3729 platform: ios, '14'
3730 install! 'cocoapods', : deterministic_uuids => false
3731
3732 target 'AlertaCivil' do
3733   use_expo_modules!
3734   post_integrate do | installer |
3735   begin
3736       expo_patch_react_imports!(installer)
3737     rescue => e
3738 Pod:: UI.warn e
3739 end
3740 end
3741 use_frameworks!
3742
3743 permissions_path = '../node_modules/react-native-permissions/ios'
3744   pod 'Permission-Contacts', : path => "#{permissions_path}/Contacts"
3745   pod 'Permission-LocationAccuracy', : path => "#{permissions_path}/
         LocationAccuracy"
3746   pod 'Permission-LocationAlways', : path => "#{permissions_path}/
         LocationAlways"
3747   pod 'Permission-LocationWhenInUse', : path => "#{permissions_path}/
         LocationWhenInUse"
3748   pod 'Permission-Notifications', : path => "#{permissions_path}/
         Notifications"
3749   pod 'react-native-contacts', : path => '../node_modules/react-native-
         contacts'
3750
3751 config = use_native_modules!
3752
3753   # Flags change depending on the env values.
3754   flags = get_default_flags()
3755 use_react_native!(
3756     : path => config[: reactNativePath],
3757     # to enable hermes on iOS, change`false` to`true` and then install
```

```
        pods
    : hermes_enabled => true,
    : fabric_enabled => flags[: fabric_enabled],
    : flipper_configuration => FlipperConfiguration.disabled,
    # An absolute path to your application root.
    : app_path => "#{Pod::Config.instance.installation_root}/.."
)

  target 'AlertaCivilTests' do
  inherit! : complete
    # Pods for testing
  end

  def fix_config(config)
    # https://github.com/CocoaPods/CocoaPods/issues/8891
if config.build_settings['DEVELOPMENT_TEAM'].nil ?
  config.build_settings['DEVELOPMENT_TEAM'] = '4Z6E862WH7'
    end
if config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'].to_f < 14
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '14'
end
end

  post_install do | installer |
  installer.generated_projects.each do | project |
    project.build_configurations.each do | config |
      fix_config(config)
      end
      project.targets.each do | target |
  target.build_configurations.each do | config |
    fix_config(config)
        end
      end
end
react_native_post_install(installer)
__apply_Xcode_12_5_M1_post_install_workaround(installer)
end
end



\end{ apendicesenv }
% ---
```