UNIVERSIDADE DE UBERABA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

THIAGO BARBOSA RIBEIRO

CAUANA SALVIANO CONCEIÇÃO

JOÃO VINICIUS BATISTA DE OLIVEIRA

DOCUMENTAÇÃO TÉCNICA DO APLICATIVO SOLIDÁRIO

UBERABA

2024

THIAGO BARBOSA RIBEIRO
CAUANA SALVIANO CONCEIÇÃO
JOÃO VINICIUS BATISTA DE OLIVEIRA

DOCUMENTAÇÃO TÉCNICA DO APLICATIVO SOLIDÁRIO

Documentação técnica do aplicativo facilitador de doações para ONGs Solidário apresentado para a disciplina Projetos Integrados II no curso de graduação em Análise e Desenvolvimentos de Sistemas da Universidade de Uberaba.

Orientador: Prof. Dr. Leonardo Campos de Assis

UBERABA
2024

1 APLICATIVO SOLIDÁRIO

O Solidário, aplicativo facilitador de doações, busca aproximar o usuário de Organizações Não Sociais (ONGs) e campanhas específicas. Desenvolvido no Android Studio em linguagem Java, suas principais funcionalidades foram implementadas para que o usuário visualize ONGs por distância, itens recebidos ou campanhas disponíveis, aumentando a assertividade das doações e ampliando a visibilidade dessas organizações.

O aplicativo utiliza uma combinação de tecnologias, como Firebase Authentication, SQLite e Google Maps, e esta documentação aborda desde a modelagem de dados e armazenamento local até a autenticação e integração com APIs externas, explorando os principais aspectos técnicos.

2 ASPECTOS TÉCNICOS

Os aspectos técnicos detalham as principais tecnologias utilizadas no aplicativo Solidário. Sua arquitetura foi projetada para ser robusta e eficiente, combinando recursos *on-line* e *off-line* para oferecer uma experiência de usuário fluida e acessível.

O armazenamento de dados locais é gerenciado pelo SQLite, que permite manter informações essenciais, como dados de ONGs, campanhas e categorias, disponíveis mesmo sem conexão à internet. Isso é complementado pelo Firebase Authentication, que assegura uma autenticação rápida e segura por meio de *login* com *e-mail* e senha ou com conta Google.

A integração com o Firebase possibilita que cada usuário tenha um identificador único (UID), utilizado tanto para a personalização da experiência quanto para o gerenciamento de permissões e dados.

3 DIAGRAMAS UML

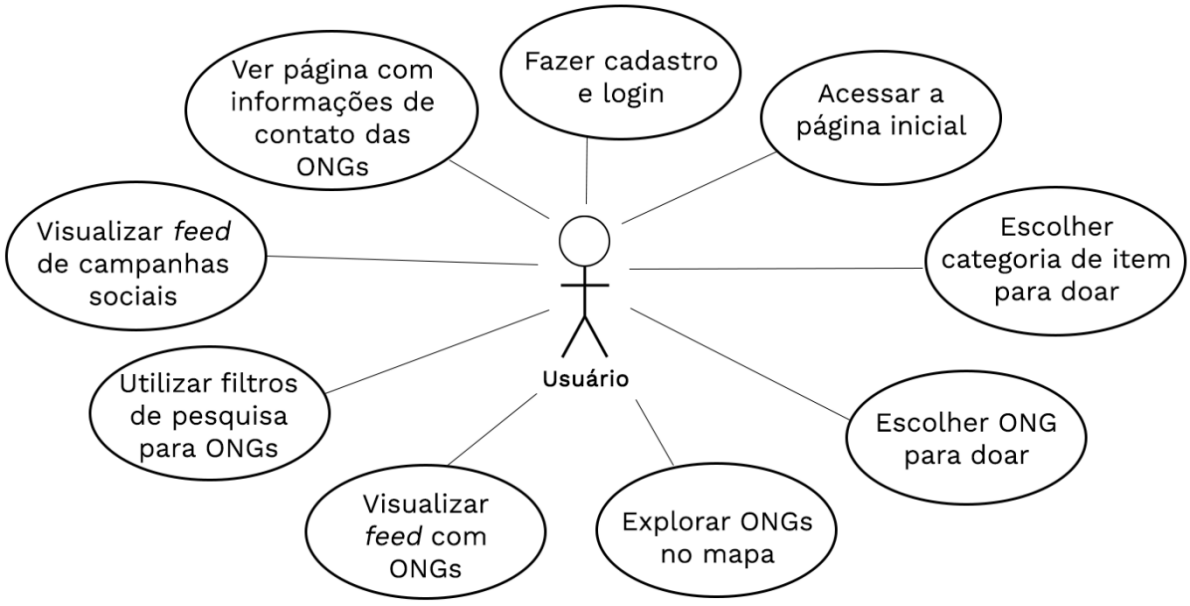
A Unified Modeling Language (UML) foi utilizada para representar visualmente o comportamento e a estrutura do sistema, facilitando a especificação dos componentes e funcionalidades do aplicativo Solidário. Foram criados diagramas de

caso de uso, sequência e classes para auxiliar na definição de requisitos, detalhar fluxos de interação e padronizar as classes e seus relacionamentos.

3.1 Diagrama de Caso de Uso

O diagrama de caso de uso (Figura 1) é uma representação visual que mostra as interações entre os usuários e o sistema, destacando as principais funcionalidades do ponto de vista do usuário. Permite definir os requisitos e visualizar como o sistema atenderá às necessidades dos usuários.

Figura 1 – Diagrama UML de caso de uso.

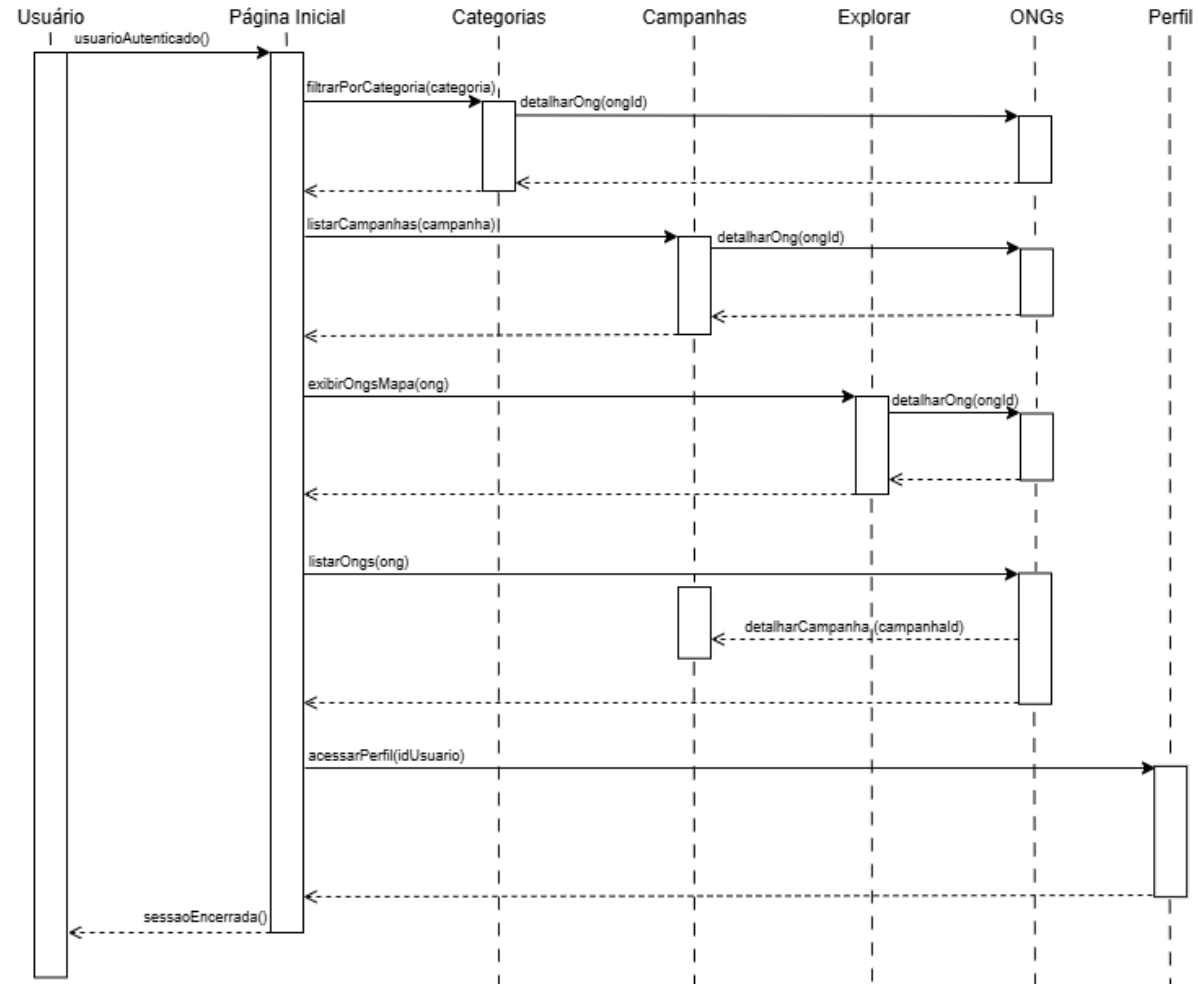


Fonte: Elaborada pelos autores (2024).

3.2 Diagrama de Sequência

O diagrama de sequência (Figura 2) descreve a ordem das interações entre objetos em um cenário específico. Ele foca no fluxo de interações, demonstrando como as operações acontecem em sequência para atingir um determinado objetivo. É fundamental para entender a lógica e o fluxo de controle de funcionalidades complexas dentro do sistema.

Figura 2 – Diagrama UML de sequência.

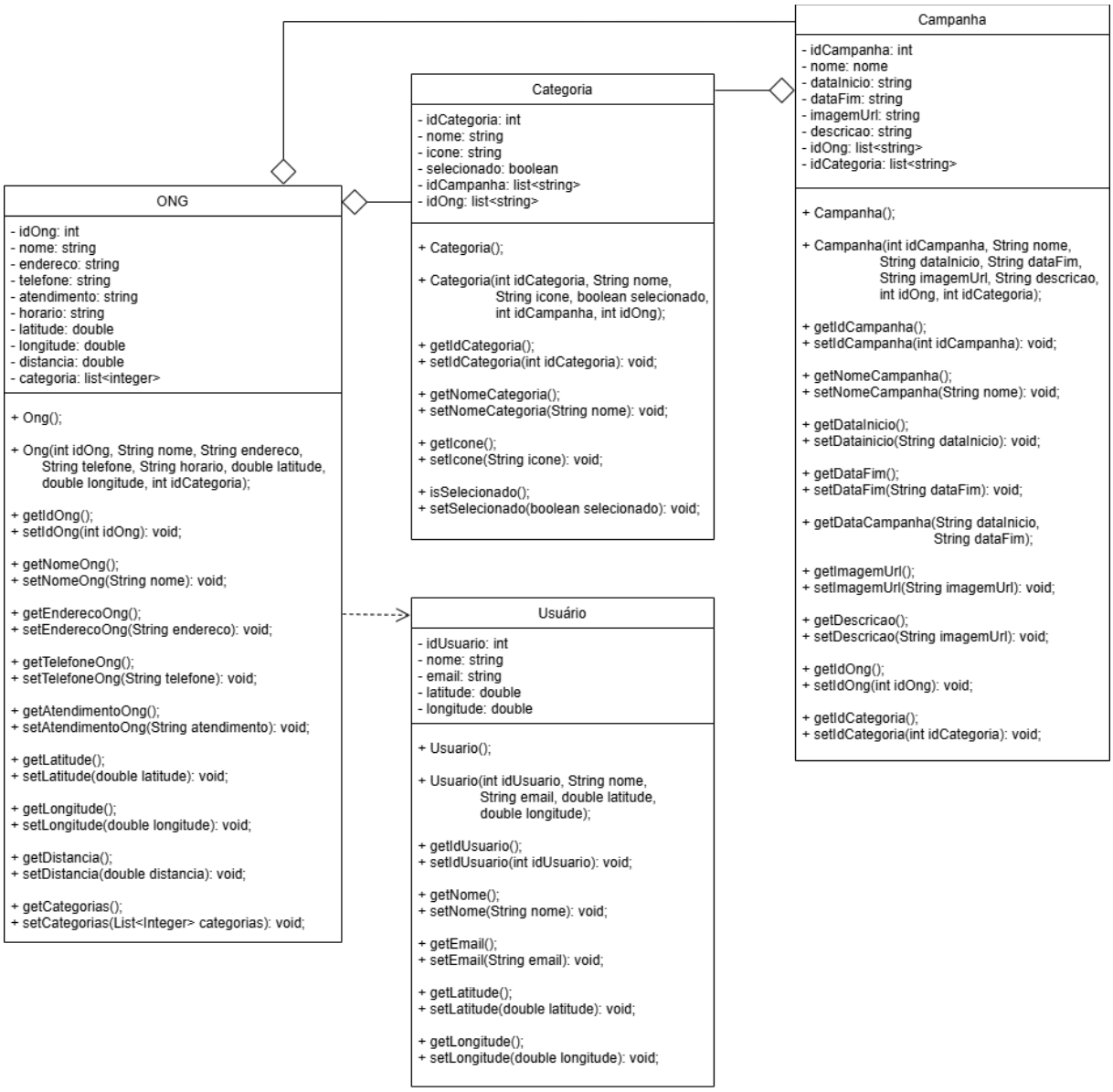


Fonte: Elaborada pelos autores (2024).

3.3 Diagrama de Classes

O diagrama de classes (Figura 3) é uma estrutura fundamental que modela os objetos de um sistema, seus atributos, métodos e os relacionamentos entre eles, permitindo uma visão abrangente da arquitetura do aplicativo. Ele foi elaborado utilizando o software *Draw.io*, uma ferramenta versátil para criar representações visuais de projetos. O diagrama detalha a estrutura estática do sistema, garantindo que os elementos estejam organizados conforme os requisitos funcionais e não funcionais definidos durante a análise de requisitos. Além disso, ele facilita a compreensão das interações entre as classes, promovendo um desenvolvimento mais estruturado e eficiente. Essa representação é essencial para validar a implementação e garantir que todas as dependências e associações estejam corretamente configuradas, alinhando-se ao propósito do aplicativo.

Figura 3 – Diagrama UML de classe.



Fonte: Elaborada pelos autores (2024).

4 SQLITE

O SQLite foi utilizado como solução de armazenamento local, oferecendo um sistema de banco de dados relacional integrado ao Android SDK, que permite um armazenamento eficiente em um único arquivo e facilita o gerenciamento de dados com baixo consumo de memória.

Embora o SQLite apresente limitações em termos de concorrência e escalabilidade, ele se destaca pela simplicidade e pela alta performance em operações *off-line*, sendo ideal para aplicativos móveis de pequeno a médio porte.

4.1 Tabelas

As tabelas formam a base do banco de dados e armazenam as informações fundamentais para o funcionamento do aplicativo. Cada tabela é criada em linguagem Structured Query Language (SQL), com comandos específicos para definir, manipular, selecionar e controlar os dados contidos em cada campo, o que possibilita pesquisas pelo nome da organização ou filtragem por categoria e tipos de itens para doação.

4.1.1 ONG

A tabela ONG armazena as informações das organizações cadastradas. Cada registro contém dados de identificação, localização e associação com campanhas ou categorias.

- **id_ong**: Chave primária da tabela com identificação única de cada ONG.
- **nome_ong**: Nome da organização, obrigatório para exibição no aplicativo.
- **endereco_ong**: Endereço físico da ONG, no formato Rua Exemplo, 123.
- **telefone_ong**: DDD (Discagem Direta à Distância) e número de telefone da ONG.
- **atendimento_ong**: Dias de atendimento (Segunda a Sexta-feira).
- **horario_ong**: Horário de funcionamento (13h às 18h).
- **latitude** e **longitude**: Coordenadas geográficas obtidas no Google Maps para exibição no mapa interativo e cálculo de distância com o usuário.
- **id_campanha**: Chave estrangeira que relaciona a ONG às campanhas diretamente na tabela associativa **campanha_ong**.
- **id_categoria**: Chave estrangeira que relaciona ONGs e categorias na tabela associativa **ong_categoria**.
- **id_usuario**: Chave estrangeira que permite identificar o usuário responsável pelo cadastro.

4.1.2 Categoria

A tabela Categoria organiza os tipos de itens aceitos pelas ONGs, permitindo aos usuários filtrarem campanhas e organizações com base em suas necessidades ou interesses.

- **id_categoria**: Chave primária da tabela referente a identificação única de cada categoria.
- **nome_categoria**: Nome descritivo da categoria, como "Alimentos", "Roupas" ou "Higiene pessoal".
- **icone_categoria**: Nome do ícone exibido visualmente em cada categoria. Seu tipo utiliza texto (*string* ou *VARCHAR*) e faz referência ao arquivo no formato Portable Network Graphic (PNG) disponível na pasta local *drawable* no projeto.
- **selecionado**: Campo numérico 0 ou 1 (valor padrão $= 0$). Representa uma condição lógica e pode ser interpretado como uma disjunção $p \vee q$, onde selecionado $= 1$ indica que a categoria está ativa ou selecionada em filtros e, selecionado $= 0$, que a categoria não está selecionada.
- **id_campanha**: Chave estrangeira que relaciona diretamente as tabelas categoria e campanha. É referenciada na tabela associativa **campanha_categoria**.
- **id_ong**: Chave estrangeira entre as tabelas categoria e campanha, referenciada na tabela associativa **ong_categoria**.

4.1.3 Campanha

A tabela Campanha armazena informações relacionadas às campanhas promovidas pelas ONGs cadastradas. Essas campanhas têm como objetivo divulgar ações específicas, como arrecadações ou eventos, e possibilitam aos usuários acessarem detalhes completos sobre elas.

- **id_campanha**: Chave primária da tabela, usada para identificar unicamente cada campanha. campanha_ong
- **nome_campanha**: Nome descritivo da campanha, como "Novembro Azul" ou "Doe Brinquedos".
- **data_campanha_inicio**: Data de início da campanha, obrigatória, utilizada para controle e exibição cronológica no aplicativo.
- **data_campanha_fim**: Data de término da campanha, opcional, permitindo campanhas de duração indeterminada.
- **imagem_url**: URL de uma imagem associada à campanha, usada para criar um design visual mais atrativo na interface.
- **descricao_campanha**: Texto detalhado que explica o propósito, objetivos e informações importantes da campanha.
- **id_ong**: Chave estrangeira que associa a campanha à ONG.
- **id_categoria**: Chave estrangeira que relaciona a campanha à categoria de itens ou serviços aceitos.

4.1.4 Usuário

A tabela Usuário armazena informações dos usuários cadastrados, permitindo a personalização da experiência e o gerenciamento de dados pessoais e de localização.

- **id_usuario**: Chave primária da tabela, representando o identificador único do usuário, gerado pelo Firebase Authentication.
- **nome_usuario**: Nome do usuário, utilizado para personalizar mensagens e exibições no aplicativo.
- **email_usuario**: Endereço de e-mail do usuário, obrigatório para login e comunicação no sistema.
- **foto_usuario**: URL da foto de perfil do usuário.

- **latitude** e **longitude**: Coordenadas geográficas da localização atual do usuário, usada para exibir ONGs e campanhas próximas no mapa interativo.

4.1.5 Redes Sociais

A tabela Redes Sociais armazena informações sobre os perfis de redes sociais das ONGs cadastradas, permitindo que os usuários sejam diretamente direcionados para a respectiva rede social.

- **id_socialmedia**: Chave primária da tabela, usada para identificar unicamente cada perfil de rede social.
- **nome_social**: Nome da rede social (por exemplo, "Facebook", "Instagram"), permitindo identificar visualmente o tipo de mídia.
- **link_social**: URL do perfil específico da ONG na rede social, direcionando o usuário para a página oficial da ONG.
- **id_ong**: Chave estrangeira que associa o perfil de rede social a uma ONG específica, garantindo que cada link esteja vinculado à ONG correta.

4.1.6 Tabelas Associativas

Para gerenciar os relacionamentos "muitos-para-muitos" entre campanhas, categorias e ONGs, foram criadas três tabelas associativas: **campanha_categoria**, **campanha_ong** e **ong_categoria**. Essas tabelas, por meio da chave estrangeira associativa, permitem que uma campanha esteja vinculada a várias categorias e ONGs, além de possibilitar que cada ONG aceite múltiplas categorias de itens ou serviços.

5 FIREBASE AUTHENTICATOR

O Firebase Authentication foi integrado ao *backend* para gerenciar o fluxo de autenticação, suportando diferentes métodos de *login*, como *e-mail* e autenticação via

Google. Configurado para oferecer uma experiência prática, o Firebase gerencia a troca de *tokens* entre o aplicativo e a API de autenticação, criando ou associando um UID único ao perfil do usuário. Esse UID é utilizado para manter as informações do usuário organizadas no banco de dados SQLite, independentemente do método de login escolhido, garantindo uma autenticação contínua e eficiente.

5.1 Fluxo de Autenticação

A autenticação permite que o usuário se identifique utilizando *e-mail* e senha ou uma conta Google. No *login* via *e-mail* e senha, o usuário insere suas credenciais, que são validadas pelo Firebase Authentication.

Se corretas, é retornado um UID único, usado para buscar informações adicionais no banco de dados local. Já no *login* com Google, o usuário é autenticado pelo Google Sign-In API, e as credenciais são enviadas ao Firebase Authentication para criar ou associar um UID.

Após a autenticação bem-sucedida, o usuário é automaticamente redirecionado para a página principal do Solidário, desde que as credenciais estejam corretas.

5.1.1 Validação de Entrada

A validação de entrada é essencial no processo de autenticação, pois garante que o *e-mail* inserido seja válido e que a senha atenda aos critérios mínimos de segurança. O método **validateInputs()**, representado pela Figura 6, verifica se o *e-mail* preenchido está no formato correto e se a senha contém os requisitos de segurança.

Caso qualquer uma dessas condições não seja atendida, é exibida uma mensagem de erro diretamente no campo correspondente, com o foco ajustado para promover uma experiência intuitiva e acessível ao usuário.

Figura 6 – Fragmentos do código, em Java, do método para validação de entrada.

```java
if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
    emailField.setError("E-mail inválido.");
    emailField.requestFocus();
    return false;
}

if (password.isEmpty()) {
    passwordField.setError("Insira a senha.");
    passwordField.requestFocus();
    return false;
}
```

Fonte: Elaborada pelos autores (2024).

5.1.2 Login com E-mail e Senha

O *login* com *e-mail* e senha permite que o Firebase Authentication faça a validação das credenciais inseridas pelo usuário. O método **signInWithEmailAndPassword()**, ilustrado na Figura 4, realiza a autenticação ao verificar o *e-mail* e a senha fornecidos.

Figura 4 – Fragmentos do código, em Java, do método para autenticação com *e-mail* e senha.

```java
public class LoginActivity extends AppCompatActivity {
    private void signInWithEmailAndPassword() { 1 usage
        mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                FirebaseUser user = mAuth.getCurrentUser();
                if (user != null) {

                    String userId = user.getUid();
                    UsuarioDAO usuarioDAO = new UsuarioDAO(new DatabaseHelper
                            ( context: this).getReadableDatabase());
                    Usuario usuario = usuarioDAO.buscarUsuarioPorId(userId);

                    if (usuario != null) {
                        Intent intent = new Intent( packageContext: LoginActivity.this,
                                DashboardActivity.class);
                        startActivity(intent);
                        finish();
                    } else {
                        //exibição e tratamento de erros
                    }
```

Fonte: Elaborada pelos autores (2024).

Após uma autenticação bem-sucedida, o UID do usuário é recuperado por meio do método **getUid()** e, em seguida, o UID é usado para buscar dados específicos do usuário armazenados localmente, utilizando a classe **UsuarioDAO**. Se os dados do usuário são encontrados no banco de dados local, a página principal do aplicativo será exibida; caso contrário, uma mensagem de falha no *login* será exibida na tela.

5.1.3 Login com Google

O *login* com Google utiliza o Google Sign-In API para autenticar o usuário, com o método **firebaseAuthWithGoogle()** representado pela Figura 5. Após a autenticação bem-sucedida no Google, o método **firebaseAuthWithGoogle()** envia as credenciais ao Firebase, que associa ou cria um UID para o usuário. Além de realizar a autenticação, o Solidário aproveita essa integração para extrair dados básicos do perfil do usuário, como nome e *e-mail*, que são armazenados localmente no banco de dados SQLite para personalizar a experiência e agilizar o acesso futuro.

Figura 5 – Fragmentos do código, em Java, do método para autenticação com Google.

```
public class LoginActivity extends AppCompatActivity {
    private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {  1 usage
        AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(),  s1: null);
        mAuth.signInWithCredential(credential).addOnCompleteListener( activity: this, task -> {
            if (task.isSuccessful()) {
                FirebaseUser user = mAuth.getCurrentUser();

                if (user != null) {
                    //lógica para armazenar dados do usuário no SQLite
                }
            } else {
                //exibição e tratamento de erros           }
            }
```

Fonte: Elaborada pelos autores (2024).

6 GOOGLE MAPS

A integração com a Google Maps API foi implementada para oferecer um mapa interativo que melhora a experiência do usuário, permitindo localizar e interagir com ONGs com base em sua proximidade geográfica. As coordenadas das ONGs

cadastradas são armazenadas no SQLite e exibidas no mapa como marcadores, de acordo com a distância do usuário.

6.2 Marcadores

Os marcadores no mapa representam as ONGs cadastradas, com base nas informações de latitude e longitude armazenadas no banco de dados SQLite. Essa abordagem garante eficiência no acesso aos dados e uma integração fluida com a interface do mapa. Cada marcador é exibido de forma personalizada, destacando a localização específica de uma ONG. Como ilustrado na Figura 6, ao clicar em um marcador, o usuário acessa uma janela informativa que exibe o nome da organização e, caso disponível, fornece um link direto para a página detalhada da ONG. Essa funcionalidade torna o aplicativo mais dinâmico e facilita o engajamento do usuário com as organizações, promovendo a interação e o fortalecimento de conexões na rede.

Figura 6 – Fragmentos do código, em Java, do método que exibe os marcadores no mapa.

```
public class ExploreActivity extends AppCompatActivity implements OnMapReadyCallback {
    private void atualizarMapaComOngs(List<Ong> ongs) {  2 usages
        googleMap.clear();

        if (ongs.isEmpty()) {
            //método para centralizar o mapa no usuário caso a lista de ONGs seja vazia
            return;
        }

        LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();
        LatLng userLocation = new LatLng(userLatitude, userLongitude);
        boundsBuilder.include(userLocation);

        // adiciona as localizações das ONGs aos marcadores
        for (Ong ong : ongs) {
            LatLng location = new LatLng(ong.getLatitude(), ong.getLongitude());
            Marker marker = googleMap.addMarker(new MarkerOptions()
                    .position(location)
                    .title(ong.getNomeOng())
                    .icon(BitmapDescriptorFactory.defaultMarker
                            (BitmapDescriptorFactory.HUE_AZURE)));
            markerOngMap.put(marker, ong);
            boundsBuilder.include(location);
        }
```

Fonte: Elaborada pelos autores (2024).

6.1 Cálculo de Proximidade

Utilizando coordenadas geográficas armazenadas no banco de dados SQLite e os dados fornecidos pelo GPS, o sistema calcula a distância entre o usuário e as ONGs cadastradas, aplicando a fórmula de Haversine (Figura 7). Esse cálculo é fundamental para ordenar e exibir apenas as ONGs que atendem aos critérios de proximidade definidos pelo usuário, como um raio específico de busca, utilizando os filtros, ou a listagem padrão por proximidade na exibição dos cartões informativos das organizações, tornando a experiência mais eficiente e prática.

Figura 7 – Fragmentos do código, em Java, da fórmula de Haversine.

```
private double calcularDistancia(double lat1, double lon1, double lat2, double lon2) {
    final int R = 6371;
    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2) +
            Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                    Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
}
```

Fonte: Elaborada pelos autores (2024).

7 CRONOGRAMA

ATIVIDADES	JUL.	AGO.	SET.	OUT.	NOV.	DEZ.
Estudo das bases teóricas	X					
Pesquisa bibliográfica	X	X	X	X		
Levantamento de dados	X	X	X			
Coleta de dados das ONGs			X	X		
Análise dos dados coletados				X		
Protótipo das telas de navegação		X				
Levantamento de requisitos		X	X			
Modelagem e diagramas UML		X				

Desenvolvimento			X	X	X	
Testes e ajustes			X	X	X	
Artigo científico	X	X	X	X	X	
Apresentação do projeto					X	
Documentação					X	X
Manual do usuário						X
Entrega da versão final						X

8 CONSIDERAÇÕES FINAIS

O desenvolvimento do aplicativo Solidário proporcionou a criação de uma solução tecnológica que visa facilitar o processo de doações e ampliar a visibilidade de ONGs e campanhas sociais. Este projeto integrou diferentes ferramentas e tecnologias, como Firebase Authentication, SQLite e Google Maps API, para oferecer uma experiência fluida, intuitiva e eficiente aos usuários. Ao longo da documentação técnica, foram detalhados os principais aspectos da implementação, desde a análise de requisitos até os cálculos de proximidade e exibição de marcadores no mapa.

O Solidário representa não apenas um avanço técnico, mas também um passo em direção a soluções que impactam positivamente a sociedade, promovendo a solidariedade e o engajamento social. Este trabalho reflete o aprendizado adquirido ao longo do curso de Análise e Desenvolvimento de Sistemas e a aplicação prática dos conhecimentos em um projeto que combina inovação, responsabilidade social e tecnologia.

Embora o aplicativo esteja em uma versão inicial, sua arquitetura modular e flexível permite expansões futuras, garantindo que ele continue relevante e eficiente. Por fim, este projeto destaca a importância de unir tecnologia e propósito, reafirmando o papel essencial dos sistemas digitais no apoio a iniciativas sociais.