PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
COORDERNAÇÃO ENSINO A DISTÂNCIA – CEAD
ESCOLA POLITÉCNICA E DE ARTES
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PROJETO INTEGRADOR V – A

ISAÍAS CORREIA DE MORAIS

MATRÍCULA: 1132024100768

GOIÂNIA
2024.

Sumário

1 OBJETIVO

O Projeto Integrador, tem como objetivo principal, desenvolver um aplicativo para dispositivos móveis, usando as operações CRUD (Create, Reade, Update, Delete). Utilizando um Banco de Dados Local, o aplicativo tem a função de cadastrar, consultar, alterar e excluir. O seguinte tema da aplicação será um Gerenciador de Contatos, que permite inserir nome, telefone e e-mail é depois ser salvo no banco de dados. Foi usada as seguintes Tecnologias a Linguagem de Programação Dart, o Framework Flutter, juntamente com um o Banco de Dados SQLlite, a IDE Visual Studio Code da Microsoft e a ferramenta de Designer Figma para fazer o protótipo do projeto, e foi usada para a construção do projeto a Metodologia Ágil Extreme Programming (XP).

2 METODOLOGIA EXTREME PROGRAMMING (XP)

Para o Desenvolvimento deste aplicativo, foi adotada a metodologia Exteme Programming XP, que consiste em ciclos, como este projeto foi especialmente criado individualmente, mas ele contém a metodologia XP. O ciclo contém principalmente simplicidade, feedback, coragem, respeito e comunicação que contém cinco ciclos e cada ciclo serve para um determinado problema para a resolução de algum problema, usando – se a metodologia ágil XP que é focada na melhoria de qualidade do software, é a capacitação de adaptação a mudanças rápidas ela foi desenvolvida para ajudar nas dificuldades de projetos de softwares complexos.

2.1 APLICANDO O XP NO PROJETO

1. Simplicidade: Foi focado em um designer simples, e eficiente nada de complexidade, a estrutura foi simples, intuitiva, fácil de mexer e cada requisito foi desenvolvido com base na proposta. Como Cadastro, Listagem, Edição, Exclusão. Qualquer usuário pode estar interagindo usando como criar um contato, editar contato, excluir e listar todos os contatos já cadastrados.

2. Feedback: Ao longo do desenvolvimento da aplicação, foi feito vários testes, para que não haja erros, quando se criava cada funcionalidade testava e se houvesse erros eram resolvidos, e testado novamente para que não tenha problemas futuros além de apenas o desenvolvedor testar foi a pedido de outras pessoas para testarem também.

3. Coragem: Ao desenvolver o aplicativo, como nunca teve experiência nas tecnologias Flutter e Dart foi um desafio por enfrentar problemas técnicos, que foi resolvido apenas na documentação revendo o que colocar no lugar certo, a coragem e essencial para poder enfrentar qualquer desafio e problema, principalmente também no banco de dados ao implementar no aplicativo para funcionar cada parte especifica como foi feita o front end e back end.

4. Respeito: O respeito foi direcionado ao processo de desenvolvimento também, que envolve a prática, disciplina, e o entendimento melhor de que o software precisa ser desenvolvido e respeito também pelo usuário final, que implica a desenvolver a solução que atenda às necessidades do usuário.

5. Comunicação: A comunicação e outro pilar fundamental importante no XP, embora ele seja uma metodologia ágil que é amplamente associada ao trabalho em equipe, e relevante no contexto de um projeto individual também, a comunicação foi o seguinte foi definido o que ia acontecer no projeto, o que ia ser desenvolvido passo a passo, quais tecnologias ser utilizadas e dentre outros passos a serem utilizados no projeto, mas em torno do desenvolvimento foi feita reuniões com o mentor para ser retirado dúvidas e perguntas sobre projeto.

3 TECNOLOGIAS UTILIZADAS

Ao longo do projeto, foram utilizadas várias tecnologias para se realizar o projeto para chegar à finalização e entrega final, foi usado como tecnologias principais foram utilizadas a Linguagem de Programação Dart, juntamente com o Framework Flutter, a IDE para a codificação do projeto o Visual Studio Code, a ferramenta de designer Figma para se realizar a prototipação e para armazenar os dados localmente no dispositivo e ter os dados seguro armazenados foi usado o SQLLite como Banco de Dados que é leve e muito conhecido por ser usado na maiorias dos dispositivos móveis é em Android.

3.1 Flutter

O Flutter é um framework de desenvolvimento de interfaces móveis criado pela Google, permitindo a criação de aplicativos nativos para iOS e Android com uma única base de código. Isso facilita o desenvolvimento, já que não é necessário criar códigos separados para cada sistema operacional. O Flutter possui uma documentação extensa, com tutoriais e vídeos que auxiliam na codificação. Ele permite criar interfaces modernas, bonitas e fluídas, e possui a funcionalidade Hot Reload, que agiliza a programação ao aplicar alterações sem precisar recarregar a aplicação.

3.2 Dart

Dart é uma linguagem de programação criada pela Google para desenvolver aplicativos Flutter, otimizada para interfaces ricas e performáticas. É intuitiva e fácil de aprender, com o objetivo inicial de substituir o JavaScript nos navegadores. Dart possui uma documentação extensa, permitindo o aprendizado e aprofundamento na linguagem. Além de ser usada em aplicações móveis, também pode ser aplicada no desenvolvimento de aplicativos desktop.

3.3 SQLLite

SQLite é um banco de dados relacional leve, usado em muitas aplicações móveis para armazenamento local. É simples, eficiente, e oferece ótimo desempenho para armazenar dados como nome, telefone e e-mail. Criado em C, não requer configurações complexas nem licenças pagas. Sua integração com o Flutter é fácil, bastando importar uma biblioteca para começar a armazenar dados. Embora não suporte grandes volumes de dados como outros bancos de dados, é amplamente utilizado por sua simplicidade e pela extensa documentação, sendo popular entre estudantes e profissionais.

3.4 Figma

Figma é uma ferramenta de design colaborativa baseada na web, usada para criar protótipos de interfaces e designs para aplicativos e websites. É fácil de usar e permite criar interfaces para diversas plataformas, como computadores, celulares e TVs. Além de ser acessado pela web, pode ser baixado no celular ou computador. Várias pessoas podem colaborar no mesmo projeto, tornando-o ideal para equipes. O Figma possui uma grande comunidade, tutoriais e dicas para novos usuários. É gratuito, com recursos pagos adicionais. Ele facilita o planejamento e prototipagem antes de iniciar a codificação.

3.5 Visual Studio Code

Visual Studio Code é um editor de código-fonte leve e intuitivo, suportando diversas linguagens, incluindo Dart e Flutter. Criado pela Microsoft, está disponível para Windows, Linux e macOS, e recebe atualizações frequentes. Além de Dart e Flutter, suporta outras linguagens de programação e é amplamente utilizado por profissionais e estudantes. O VS Code melhora a produtividade com recursos como autocompletar código, integração com Git e diversos plugins. Sua leveza permite ser instalado em qualquer computador, seja rápido ou mais lento, oferecendo diversas vantagens para o desenvolvimento.

3.6 Git

Git é uma ferramenta de controle de versão usada no projeto para registrar alterações no código de forma segura e organizada. Ela permite rastrear erros e manter um histórico detalhado do desenvolvimento. Suas vantagens incluem a possibilidade de reverter para versões anteriores, melhor organização com commits documentados e fácil integração com repositórios online, como o GitHub. Além disso, Git pode ser instalado no computador e controlado via terminal de comandos.

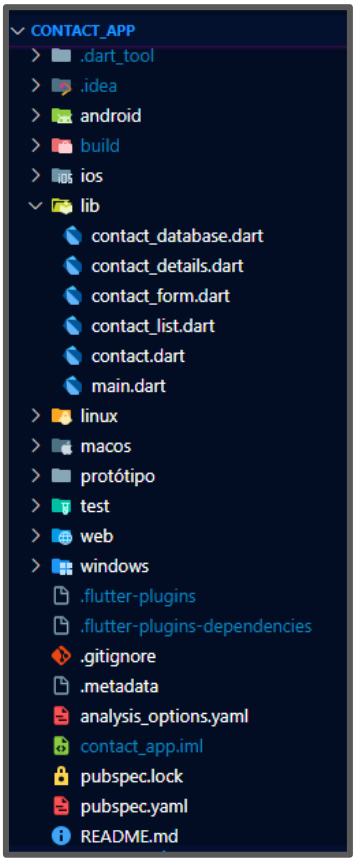
3.7 GitHub

GitHub é uma plataforma baseada na web que hospeda repositórios Git, permitindo compartilhar e colaborar em projetos de desenvolvimento de software.

Ideal para armazenar grandes códigos, também facilita a contribuição em projetos open source. Foi utilizado no projeto para armazenar o código, possibilitando acesso remoto e compartilhamento com outras pessoas, como professores e avaliadores. O GitHub permite definir projetos como privados ou públicos, oferecendo a possibilidade de acessar o código de qualquer lugar, desde que tenha o link.

4 Estrutura da Aplicação

A aplicação foi organizada de forma modular, com cada arquivo no diretório lib responsável por uma funcionalidade específica. Essa abordagem facilita o entendimento do código, a implementação de novas funcionalidades e a manutenção futura. A estrutura modular torna mais simples adicionar recursos ou realizar ajustes conforme a aplicação evolui, garantindo um desenvolvimento eficiente e organizado.



4.1 Main.Dart

```dart
import 'package:flutter/material.dart';
import 'contact_list.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'App Contatos',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const ContactListScreen(),
    );
  }
}
```

No arquivo main.dart e o ponto de entrada do aplicativo Flutter que desempenha as seguintes funções:

- Inicialização: A função main() executa o aplicativo usando o widget raiz MyApp
- Configuração Global: O widget MaterialApp define o tema da aplicação com cores baseadas no azul e remove a faixa de debug
- Tela Inicial: A tela principal ao iniciar o aplicativo é definida como Contact_List que exibe a lista de contatos.

4.2 Contact.Dart

O arquivo contact.dart define o modelo de dados para os contatos do aplicativo ele é usado para estruturar e organizar as informações de cada contato suas principais funções incluem:

Atributos do Contato:

- Id: identificador único
- Nome: nome do contato
- Telefone: Número de telefone
- Email: endereço de Email

```dart
class Contact {
  int? id;
  final String name;
  final String phone;
  final String email;

  Contact({
    this.id,
    required this.name,
    required this.phone,
    required this.email,
  });

  // Converte um objeto Contact para um Map (para o banco de dados)
  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'phone': phone,
      'email': email,
    };
  }

  // Cria um objeto Contact a partir de um Map (do banco de dados)
  factory Contact.fromMap(Map<String, dynamic> map) {
    return Contact(
      id: map['id'] as int?,
      name: map['name'] as String,
      phone: map['phone'] as String,
      email: map['email'] as String,
    );
  }

  // Método copyWith para criar uma cópia com novos valores
  Contact copyWith({
    int? id,
    String? name,
    String? phone,
    String? email,
  }) {
    return Contact(
      id: id ?? this.id,         // Se 'id' for passado, usa o novo; se não, usa o existente
      name: name ?? this.name,   // O mesmo vale para os outros campos
      phone: phone ?? this.phone,
      email: email ?? this.email,
    );
  }
}

```

Conversão de Dados:

- Possui métodos para converter os dados de um contato entre
- Um objeto Dart (usado no código do aplicativo)
- Um mapa (Map<String, dynamic>) usado pelo banco de dados SQLLite

4.3 Contact_List.Dart

lib > contact_list.dart > _ContactListScreenState > build

```dart
import 'package:flutter/material.dart';
import 'contact_form.dart';
import 'contact_database.dart';
import 'contact.dart';
import 'contact_details.dart';

class ContactListScreen extends StatefulWidget {
  const ContactListScreen({super.key});

  @override
  State<ContactListScreen> createState() => _ContactListScreenState();
}

class _ContactListScreenState extends State<ContactListScreen> {
  List<Contact> contacts = [];
  List<Contact> filteredContacts = [];
  String searchQuery = '';
  bool isLoading = true; // Controle de carregamento

  @override
  void initState() {
    super.initState();
    _loadContacts();
  }

  Future<void> _loadContacts() async {
    await Future.delayed(const Duration(seconds: 2)); // Atraso maior para o carregamento (2 segundos)
    final data = await ContactDatabase.instance.readAllContacts();
    setState(() {
      contacts = data..sort((a, b) => a.name.compareTo(b.name)); // Ordenando os contatos por nome
      filteredContacts = contacts; // Inicializa com todos os contatos
      isLoading = false; // Indica que o carregamento terminou
    });
  }
```

```dart
  }

  void _filterContacts(String query) {
    setState(() {
      searchQuery = query;
      if (query.isEmpty) {
        filteredContacts = contacts; // Restaura a lista original quando não há pesquisa
      } else {
        filteredContacts = contacts
            .where((contact) => contact.name.toLowerCase().contains(query.toLowerCase()))
            .toList();
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    // Agrupando os contatos pela primeira letra do nome
    Map<String, List<Contact>> groupedContacts = {};
    for (var contact in filteredContacts) {
      String firstLetter = contact.name[0].toUpperCase(); // Pegando a primeira letra
      if (!groupedContacts.containsKey(firstLetter)) {
        groupedContacts[firstLetter] = [];
      }
      groupedContacts[firstLetter]!.add(contact);
    }

    return Scaffold(
      appBar: AppBar(
        title: const Text('Lista de Contatos'),
        actions: [
          // Ícone de adicionar com transição personalizada
```

```dart
// Ícone de adicionar com transição personalizada
Padding(
  padding: const EdgeInsets.only(right: 8.0),
  child: GestureDetector(
    onTap: () {
      Navigator.push(
        context,
        PageRouteBuilder(
          pageBuilder: (context, animation, secondaryAnimation) => const ContactFormScreen(),
          transitionsBuilder: (context, animation, secondaryAnimation, child) {
            const begin = Offset(1.0, 0.0); // Inicia da direita
            const end = Offset.zero;
            const curve = Curves.easeInOut;
            var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
            var offsetAnimation = animation.drive(tween);

            return SlideTransition(position: offsetAnimation, child: child);
          },
        ), // PageRouteBuilder
      ).then((_) => _loadContacts());
    },
    child: Container(
      width: 31, // Define o tamanho do botão
      height: 45,
      decoration: BoxDecoration(
        color: Colors.green, // Cor do botão
        shape: BoxShape.circle, // Faz o botão ficar redondo
      ), // BoxDecoration
      child: const Icon(
        Icons.add,
        color: Colors.white, // Cor do ícone
        size: 30, // Tamanho do ícone
```

```
                    size: 30, // Tamanho do ícone
                  ), // Icon
                ), // Container
              ), // GestureDetector
            ), // Padding
            // Ícone de lupa para pesquisar
            Padding(
              padding: const EdgeInsets.only(right: 8.0),
              child: IconButton(
                icon: const Icon(Icons.search),
                onPressed: () {
                  showSearch(
                    context: context,
                    delegate: ContactSearchDelegate(
                      onQueryChanged: _filterContacts,
                      initialContacts: contacts,
                    ), // ContactSearchDelegate
                  );
                },
              ), // IconButton
            ), // Padding
          ],
        ), // AppBar
        body: isLoading
            ? Center(child: CircularProgressIndicator()) // Mostra um indicador de carregamento enquanto os dados são carregados
            : FadeTransition(
                opacity: AlwaysStoppedAnimation(1.0), // Opacidade constante para a animação suave
                child: Column(
                  children: [
                    // Mostra o número de contatos encontrados
                    Padding(
                      padding: const EdgeInsets.all(8.0),
```

```
127                     padding: const EdgeInsets.all(8.0),
128                     child: Text(
129                       '${filteredContacts.length} contatos encontrados', // Exibe o número de contatos encontrados
130                       style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
131                     ), // Text
132                   ), // Padding
133                   Expanded(
134                     child: ListView(
135                       children: groupedContacts.keys.isEmpty
136                           ? [Center(child: Text('Nenhum contato encontrado'))] // Mensagem caso não haja contatos filtrados
137                           : groupedContacts.keys.map((letter) {
138                               return Column(
139                                 crossAxisAlignment: CrossAxisAlignment.start,
140                                 children: [
141                                   // Exibe a letra do grupo
142                                   Padding(
143                                     padding: const EdgeInsets.all(8.0),
144                                     child: Text(
145                                       letter,
146                                       style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
147                                     ), // Text
148                                   ), // Padding
149                                   // Exibe os contatos do grupo
150                                   Column(
151                                     children: groupedContacts[letter]!
152                                         .map((contact) => Card(
153                                               child: ExpansionTile(
154                                                 title: Text(contact.name),
155                                                 subtitle: Text(contact.phone),
156                                                 leading: const CircleAvatar(
157                                                   backgroundColor: Colors.blue,
158                                                   child: Icon(Icons.person, color: Colors.white),
```

```
158                                   child: Icon(Icons.person, color: Colors.white),
159                                 ), // CircleAvatar
160                                 trailing: const Icon(Icons.arrow_drop_down),
161                                 children: [
162                                   Padding(
163                                     padding: const EdgeInsets.all(8.0),
164                                     child: Row(
165                                       children: [
166                                         Expanded(
167                                           child: Column(
168                                             crossAxisAlignment: CrossAxisAlignment.start,
169                                             children: [
170                                               Text('Nome: ${contact.name}', style: const TextStyle(fontSize: 16)),
171                                               const SizedBox(height: 8),
172                                               Text('Telefone: ${contact.phone}', style: const TextStyle(fontSize: 16)),
173                                               const SizedBox(height: 8),
174                                               Text('E-mail: ${contact.email}', style: const TextStyle(fontSize: 16)),
175                                             ],
176                                           ), // Column
177                                         ), // Expanded
178                                         IconButton(
179                                           icon: const Icon(Icons.info, color: Colors.grey),
180                                           iconSize: 35,
181                                           onPressed: () {
182                                             Navigator.push(
183                                               context,
184                                               MaterialPageRoute(
185                                                 builder: (context) => ContactDetailsScreen(contact: contact),
186                                               ), // MaterialPageRoute
187                                             ).then((_) => _loadContacts());
188                                           },
189                                         ), // IconButton
```

```
                                                                ), // IconButton
                                                              ],
                                                            ), // Row
                                                          ), // Padding
                                                        ],
                                                      ), // ExpansionTile
                                                    )) // Card
                                                .toList(),
                                            ), // Column
                                          ],
                                        ); // Column
                                      }).toList(),
                                ), // ListView
                              ), // Expanded
                            ],
                          ), // Column
                        ), // FadeTransition
                    ); // Scaffold
                  }
                }

                class ContactSearchDelegate extends SearchDelegate<String> {
                  final Function(String) onQueryChanged;
                  final List<Contact> initialContacts;

                  ContactSearchDelegate({
                    required this.onQueryChanged,
                    required this.initialContacts,
                  });

                  @override
                  String get searchFieldLabel => 'Buscar Contatos...';
```

```
220     String get searchFieldLabel => 'Buscar Contatos...';
221
222     @override
223     List<Widget>? buildActions(BuildContext context) {
224       return [
225         IconButton(
226           icon: const Icon(Icons.clear),
227           onPressed: () {
228             query = '';
229             onQueryChanged(query); // Atualiza a lista de contatos
230           },
231         ), // IconButton
232       ];
233     }
234
235     @override
236     Widget? buildLeading(BuildContext context) {
237       return IconButton(
238         icon: const Icon(Icons.arrow_back),
239         onPressed: () {
240           close(context, ''); // Fecha a busca
241           onQueryChanged(''); // Restaura a lista de contatos sem filtro
242         },
243       ); // IconButton
244     }
245
246     @override
247     Widget buildResults(BuildContext context) {
248       if (query.isEmpty) {
249         return Center(child: Text('Encontre algum Contato'));
250       }
251
```

```
251
252       final results = initialContacts
253           .where((contact) => contact.name.toLowerCase().contains(query.toLowerCase()))
254           .toList();
255
256       return results.isEmpty
257           ? Center(child: Text('Nenhum contato encontrado'))
258           : Column(
259               children: [
260                 Padding(
261                   padding: const EdgeInsets.all(8.0),
262                   child: Text(
263                     '${results.length} contatos encontrados', // Exibe o número de contatos encontrados
264                     style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
265                   ), // Text
266                 ), // Padding
267                 Expanded(
268                   child: ListView(
269                     children: results.map((contact) {
270                       return Card(
271                         child: ExpansionTile(
272                           title: Text(contact.name),
273                           subtitle: Text(contact.phone),
274                           leading: const CircleAvatar(
275                             backgroundColor: Colors.blue,
276                             child: Icon(Icons.person, color: Colors.white),
277                           ), // CircleAvatar
278                           trailing: const Icon(Icons.arrow_drop_down),
279                           children: [
280                             Padding(
281                               padding: const EdgeInsets.all(8.0),
282                               child: Row(
```

```
                        child: Row(
                          children: [
                            Expanded(
                              child: Column(
                                crossAxisAlignment: CrossAxisAlignment.start,
                                children: [
                                  Text('Nome: ${contact.name}', style: const TextStyle(fontSize: 16)),
                                  const SizedBox(height: 8),
                                  Text('Telefone: ${contact.phone}', style: const TextStyle(fontSize: 16)),
                                  const SizedBox(height: 8),
                                  Text('E-mail: ${contact.email}', style: const TextStyle(fontSize: 16)),
                                ],
                              ), // Column
                            ), // Expanded
                            IconButton(
                              icon: const Icon(Icons.info, color: Colors.grey),
                              iconSize: 35,
                              onPressed: () {
                                Navigator.push(
                                  context,
                                  MaterialPageRoute(
                                    builder: (context) => ContactDetailsScreen(contact: contact),
                                  ), // MaterialPageRoute
                                ).then((_) => onQueryChanged('')); // Restaura a lista de contatos
                              },
                            ), // IconButton
                          ],
                        ), // Row
                      ), // Padding
                    ],
                  ), // ExpansionTile
                ); // Card
```

```
                    ); // Card
                }).toList(),
              ), // ListView
            ), // Expanded
          ],
        ); // Column
  }

  @override
  Widget buildSuggestions(BuildContext context) {
    if (query.isEmpty) {
      return Center(child: Text('Encontre algum Contato'));
    }

    final suggestions = initialContacts
        .where((contact) => contact.name.toLowerCase().contains(query.toLowerCase()))
        .toList();

    return suggestions.isEmpty
        ? Center(child: Text('Nenhum contato encontrado'))
        : Column(
            children: [
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: Text(
                  '${suggestions.length} contatos encontrados', // Exibe o número de contatos encontrados nas sugestões
                  style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
                ), // Text
              ), // Padding
              Expanded(
                child: ListView(
                  children: suggestions.map((contact) {
```

```
344                   children: suggestions.map((contact) {
345                     return Card(
346                       child: ExpansionTile(
347                         title: Text(contact.name),
348                         subtitle: Text(contact.phone),
349                         leading: const CircleAvatar(
350                           backgroundColor: Colors.blue,
351                           child: Icon(Icons.person, color: Colors.white),
352                         ), // CircleAvatar
353                         trailing: const Icon(Icons.arrow_drop_down),
354                         children: [
355                           Padding(
356                             padding: const EdgeInsets.all(8.0),
357                             child: Row(
358                               children: [
359                                 Expanded(
360                                   child: Column(
361                                     crossAxisAlignment: CrossAxisAlignment.start,
362                                     children: [
363                                       Text('Nome: ${contact.name}', style: const TextStyle(fontSize: 16)),
364                                       const SizedBox(height: 8),
365                                       Text('Telefone: ${contact.phone}', style: const TextStyle(fontSize: 16)),
366                                       const SizedBox(height: 8),
367                                       Text('E-mail: ${contact.email}', style: const TextStyle(fontSize: 16)),
368                                     ],
369                                   ), // Column
370                                 ), // Expanded
371                                 IconButton(
372                                   icon: const Icon(Icons.info, color: Colors.grey),
373                                   iconSize: 35,
374                                   onPressed: () {
375                                     Navigator.push(
```

```
375                                     Navigator.push(
376                                       context,
377                                       MaterialPageRoute(
378                                         builder: (context) => ContactDetailsScreen(contact: contact),
379                                       ), // MaterialPageRoute
380                                     ).then((_) => onQueryChanged('')); // Restaura a lista de contatos
381                                   },
382                                 ), // IconButton
383                               ],
384                             ), // Row
385                           ), // Padding
386                         ],
387                       ), // ExpansionTile
388                     ); // Card
389                   }).toList(),
390                 ), // ListView
391               ), // Expanded
392             ],
393           ); // Column
394     }
395   }
396
```

O arquivo contact_list.dart é responsável por exibir a lista de contatos cadastrados no aplicativo ela tem as seguintes funções principais:

Exibição da Lista:

- Utiliza um widget como ListView para apresentar todos os contatos armazenados no banco de dados
- Cada item da lista é representado de forma simples, como um card ou linha com as informações básicas do contato

Interatividade:

- Permite ao usuário interagir com a lista, podendo selecionar um contato para visualizar ou editar seus detalhes

Conexão com o Banco de Dados:

- Ao carregar a lista, o arquivo busca os dados no banco de dados e os exibe dinamicamente

4.4 Contact_Form.Dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_masked_text2/flutter_masked_text2.dart'; // Para a máscara do telefone
import 'contact_database.dart';
import 'contact.dart';

class ContactFormScreen extends StatefulWidget {
  final Contact? contact;

  const ContactFormScreen({super.key, this.contact});

  @override
  State<ContactFormScreen> createState() => _ContactFormScreenState();
}

class _ContactFormScreenState extends State<ContactFormScreen> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _phoneController = MaskedTextController(mask: '(00)00000-0000'); // Máscara para telefone
  final _emailController = TextEditingController();

  @override
  void initState() {
    super.initState();
    if (widget.contact != null) {
      _nameController.text = widget.contact!.name;
      _phoneController.text = widget.contact!.phone;
      _emailController.text = widget.contact!.email;
    }
  }

  void _saveContact() async {
    if (_formKey.currentState!.validate()) {
      final newContact = Contact(
        name: _nameController.text,
        phone: _phoneController.text,
        email: _emailController.text, // E-mail estará validado
      );

      if (widget.contact == null) {
        await ContactDatabase.instance.createContact(newContact);
      } else {
        newContact.id = widget.contact!.id;
        await ContactDatabase.instance.updateContact(newContact);
      }

      Navigator.pop(context);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.contact == null ? 'Novo Contato' : 'Editar Contato'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                controller: _nameController,
                decoration: const InputDecoration(labelText: 'Nome'),
                validator: (value) =>
                    value == null || value.isEmpty ? 'Informe o nome' : null,
              ),
              TextFormField(
                controller: _phoneController,
                keyboardType: TextInputType.phone,
                decoration: const InputDecoration(labelText: 'Telefone'),
                validator: (value) =>
                    value == null || value.isEmpty ? 'Informe o telefone' : null,
              ),
              TextFormField(
                controller: _emailController,
                decoration: const InputDecoration(labelText: 'E-mail'),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Informe o e-mail';
                  }
                  // Expressão regular para verificar o formato de e-mail
                  String pattern =
                      r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$';
                  RegExp regExp = RegExp(pattern);
                  if (!regExp.hasMatch(value)) {
                    return 'Informe um e-mail válido (exemplo: exemplo@dominio.com)';
                  }
                  return null;
                },
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: _saveContact,
                child: const Text('Salvar'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

O arquivo contact_form.dart e resposavel pela tela de criação e edição de contatos suas principais funções incluem

Formulário de Entrada:

- Contém campos de texto para o usuário inserir ou editar informações de um contato, como nome, telefone e e-mail.

Validação de Dados:

- Implementa a validação dos dados inseridos, garantindo que o usuário forneça informações no formato correto antes de salvar

Ações de Criação e Edição:

- Dependendo do contexto (criação ou edição), o formulário salva as informações no banco de dados atualizando ou criando um contato

4.5 Contact_Details.Dart

```dart
import 'package:flutter/material.dart';
import 'contact_database.dart';
import 'contact.dart';
import 'contact_form.dart';  // Para editar o contato

class ContactDetailsScreen extends StatelessWidget {
  final Contact contact;

  const ContactDetailsScreen({super.key, required this.contact});

  // Função para excluir o contato
  void _deleteContact(BuildContext context) async {
    bool? confirmDelete = await showDialog<bool>(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: const Text('Excluir Contato'),
          content: Text(
            'Você tem certeza que deseja excluir o contato "${contact.name}"?\nTelefone: ${contact.phone}\nE-mail: ${contact.email}',
          ),
          actions: <Widget>[
            TextButton(
              child: const Text('Cancelar'),
              onPressed: () {
                Navigator.of(context).pop(false); // Não excluir
              },
            ),
            TextButton(
              child: const Text('Excluir'),
              onPressed: () {
                Navigator.of(context).pop(true); // Confirmou a exclusão
              },
            ),
          ],
        );
      },
    );

    if (confirmDelete == true) {
      await ContactDatabase.instance.deleteContact(contact.id!);
      Navigator.pop(context); // Volta para a tela anterior após excluir
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Detalhes do Contato'),
        backgroundColor: const Color.fromARGB(255, 255, 255, 255),  // Cor personalizada da AppBar
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Nome: ${contact.name}', style: const TextStyle(fontSize: 18)),
            Text('Telefone: ${contact.phone}', style: const TextStyle(fontSize: 18)),
            Text('E-mail: ${contact.email}', style: const TextStyle(fontSize: 18)),
            const SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                // Botão de editar com ícone abaixo
                Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text('Editar'),
                    IconButton(
                      icon: const Icon(Icons.edit, color: Colors.blue),
                      onPressed: () {
                        // Navega para a tela de edição
                        Navigator.push(
                          context,
                          MaterialPageRoute(
                            builder: (context) => ContactFormScreen(contact: contact),
                          ),
                        );
                      },
                    ),
                  ],
                ),
                // Botão de excluir com ícone abaixo
                Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text('Excluir'),
                    IconButton(
                      icon: const Icon(Icons.delete, color: Colors.red),
                      onPressed: () => _deleteContact(context),  // Excluir o contato
                    ),
                  ],
                ),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

O arquivo contact_details.dart é responsável pela tela que exibe os detalhes completos de um contato selecionado. Suas principais funções incluem:

Exibição de Informações:

- Exibe os dados completos de um contato, como nome, telefone e e-mail, de forma detalhada.

Visualização Simples:

- Permite ao usuário visualizar as informações de um contato de maneira clara e sem a necessidade de editar os dados.

Acesso a Detalhes:

- Este arquivo é acessado quando o usuário seleciona um contato da lista, proporcionando uma visão mais completa das informações desse contato.

4.6 Contact_DataBase.Dart

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'contact.dart';

class ContactDatabase {
  static final ContactDatabase instance = ContactDatabase._init();

  static Database? _database;

  ContactDatabase._init();

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB('contacts.db');
    return _database!;
  }

  Future<Database> _initDB(String filePath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filePath);

    return await openDatabase(
      path,
      version: 1,
      onCreate: _createDB,
      onUpgrade: _onUpgrade, // Adicionando um método de upgrade
    );
  }

  Future _createDB(Database db, int version) async {
    const idType = 'INTEGER PRIMARY KEY AUTOINCREMENT';
    const textType = 'TEXT NOT NULL';

    await db.execute('''
      CREATE TABLE contacts (
        id $idType,
        name $textType,
        phone $textType,
        email $textType
      )
    ''');
  }

  Future _onUpgrade(Database db, int oldVersion, int newVersion) async {
    // Essa função é chamada para atualizações do banco
    if (oldVersion < newVersion) {
      // Ações para upgrade do banco de dados, se necessário
    }
  }

  Future<Contact> createContact(Contact contact) async {
    final db = await instance.database;
    final id = await db.insert('contacts', contact.toMap());
    return contact.copyWith(id: id);
  }

  Future<Contact?> readContact(int id) async {
    final db = await instance.database;
    final maps = await db.query(
      'contacts',
      columns: ['id', 'name', 'phone', 'email'],
      where: 'id = ?',
      whereArgs: [id],
    );

    if (maps.isNotEmpty) {
      return Contact.fromMap(maps.first);
    } else {
      return null;
    }
  }

  Future<List<Contact>> readAllContacts() async {
    final db = await instance.database;
    const orderBy = 'name ASC';
    final result = await db.query('contacts', orderBy: orderBy);
    return result.map((map) => Contact.fromMap(map)).toList();
  }

  Future<int> updateContact(Contact contact) async {
    final db = await instance.database;
    return db.update(
      'contacts',
      contact.toMap(),
      where: 'id = ?',
      whereArgs: [contact.id],
    );
  }

  Future<int> deleteContact(int id) async {
    final db = await instance.database;
    return db.delete(
      'contacts',
      where: 'id = ?',
      whereArgs: [id],
    );
  }

  Future close() async {
    final db = await instance.database;
    db.close();
  }
}

```

O arquivo contact_database.dart é responsável por gerenciar a interação com o banco de dados SQLite, onde as informações dos contatos são armazenadas. Suas principais funções incluem:

Gerenciamento do Banco de Dados:

- Cria e configura o banco de dados SQLite.
- Define a estrutura da tabela contacts com os campos necessários (como id, nome, telefone e email).

Operações CRUD:

- Implementa as funções para realizar operações CRUD (Create, Read, Update, Delete) no banco de dados.
- Permite adicionar, consultar, editar e excluir contatos de forma eficiente.

Conexão com o Banco de Dados:

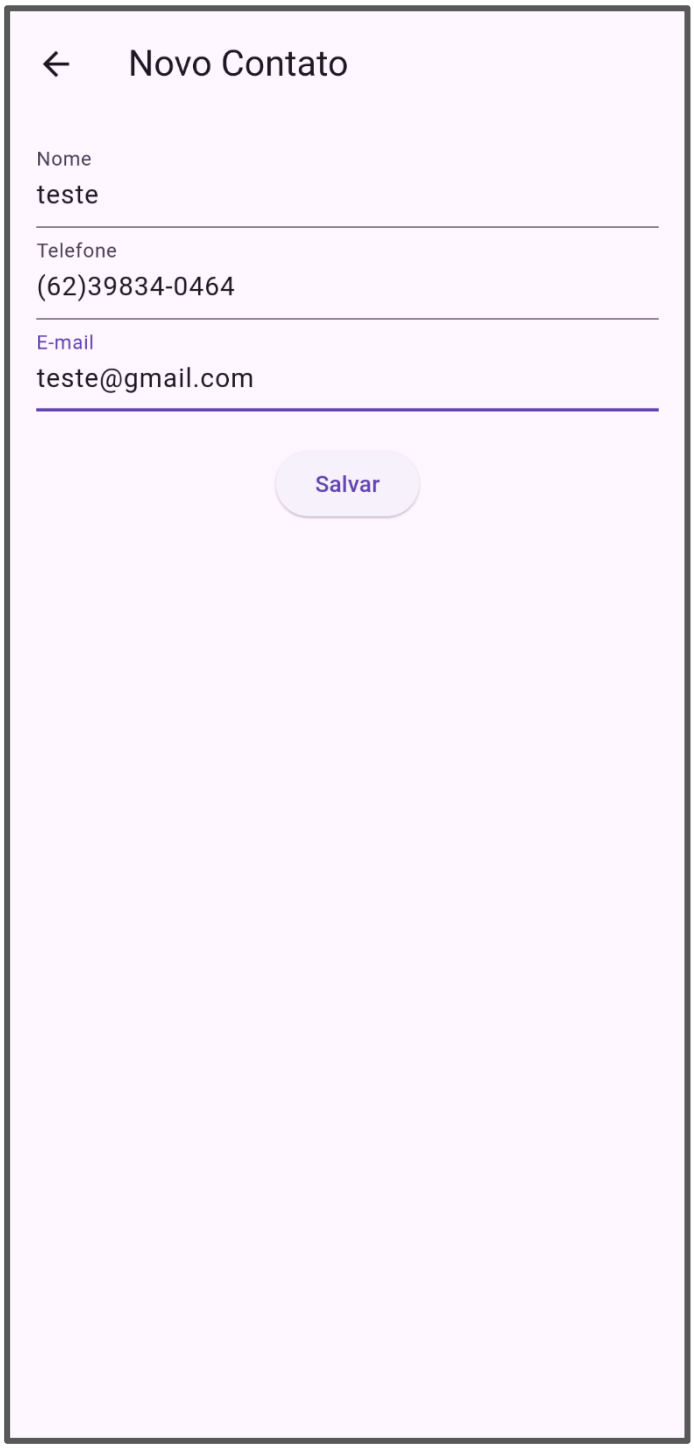
- Garante que a aplicação esteja conectada ao banco de dados local e que as operações sejam realizadas corretamente.

5 Implementação das Funcionalidades de CRUD

Foram implementadas as funcionalidades CRUD (Create, Read, Update e Delete) no banco de dados SQLite, garantindo a persistência dos dados dos contatos. Abaixo está a descrição das implementações realizadas:

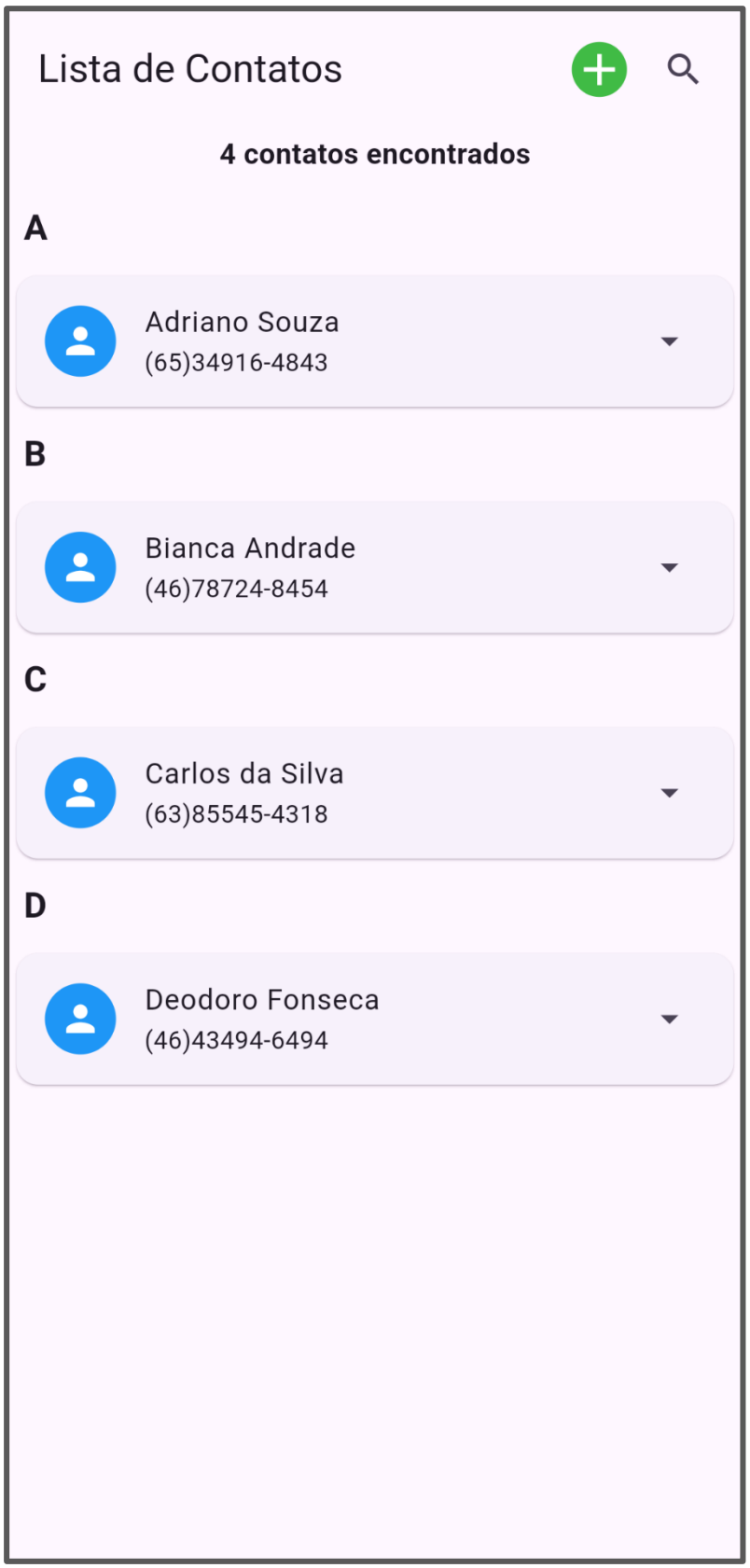
1. Create (Criar):

Quando o usuário cria um contato no aplicativo, os dados inseridos (Nome, Telefone e Email) são diretamente armazenados no banco de dados, na tabela **contacts**. Essa funcionalidade garante que as informações do contato sejam persistidas de forma eficiente e acessível para futuras consultas, edições ou exclusões.



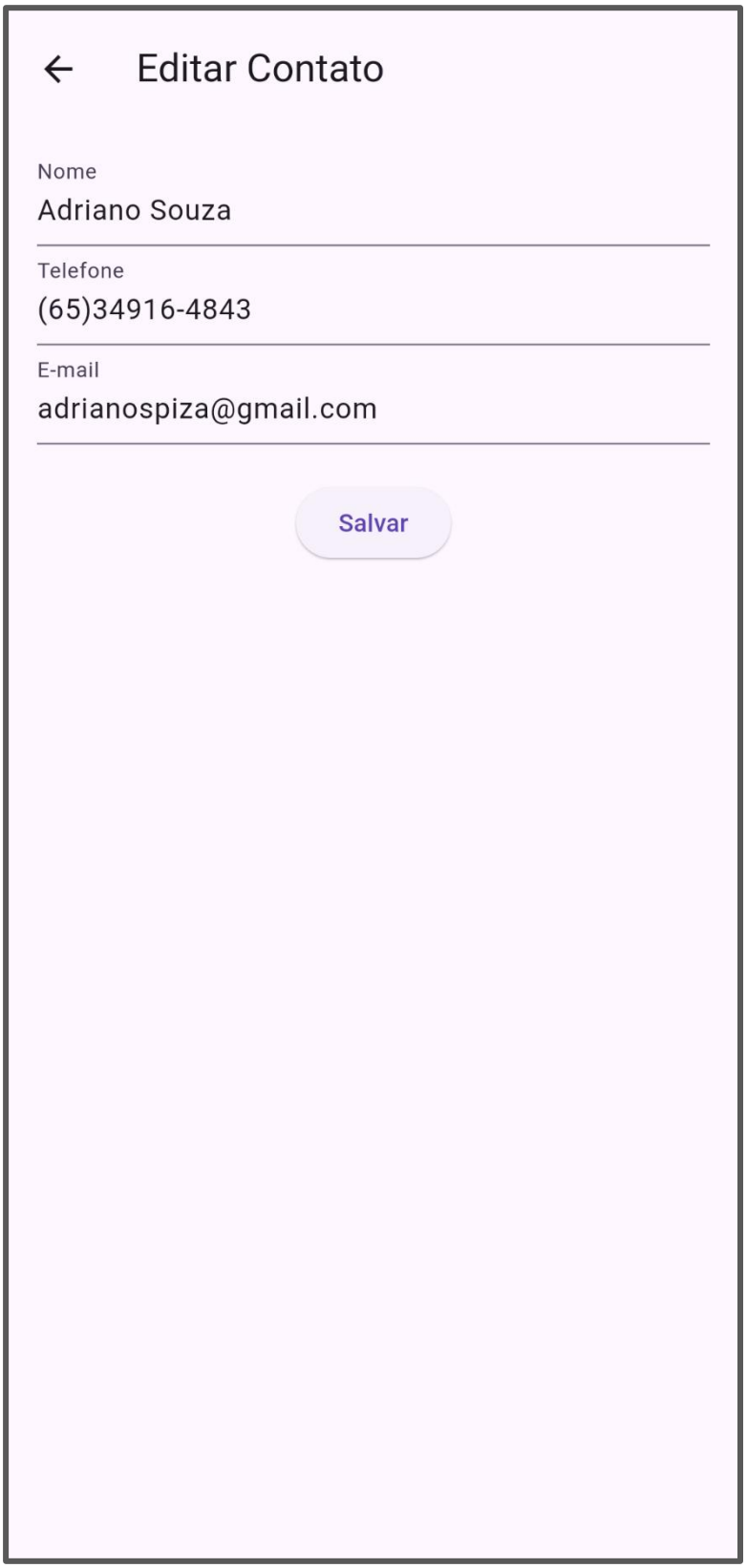
2. Read (Ler):

A operação de leitura é realizada sempre que a lista de contatos é exibida. O aplicativo consulta o banco de dados utilizando o comando SQL **SELECT** para recuperar todos os contatos armazenados. Os dados são apresentados na interface de forma organizada, permitindo que o usuário visualize a lista e, ao selecionar um contato, detalhe as informações completas do registro.



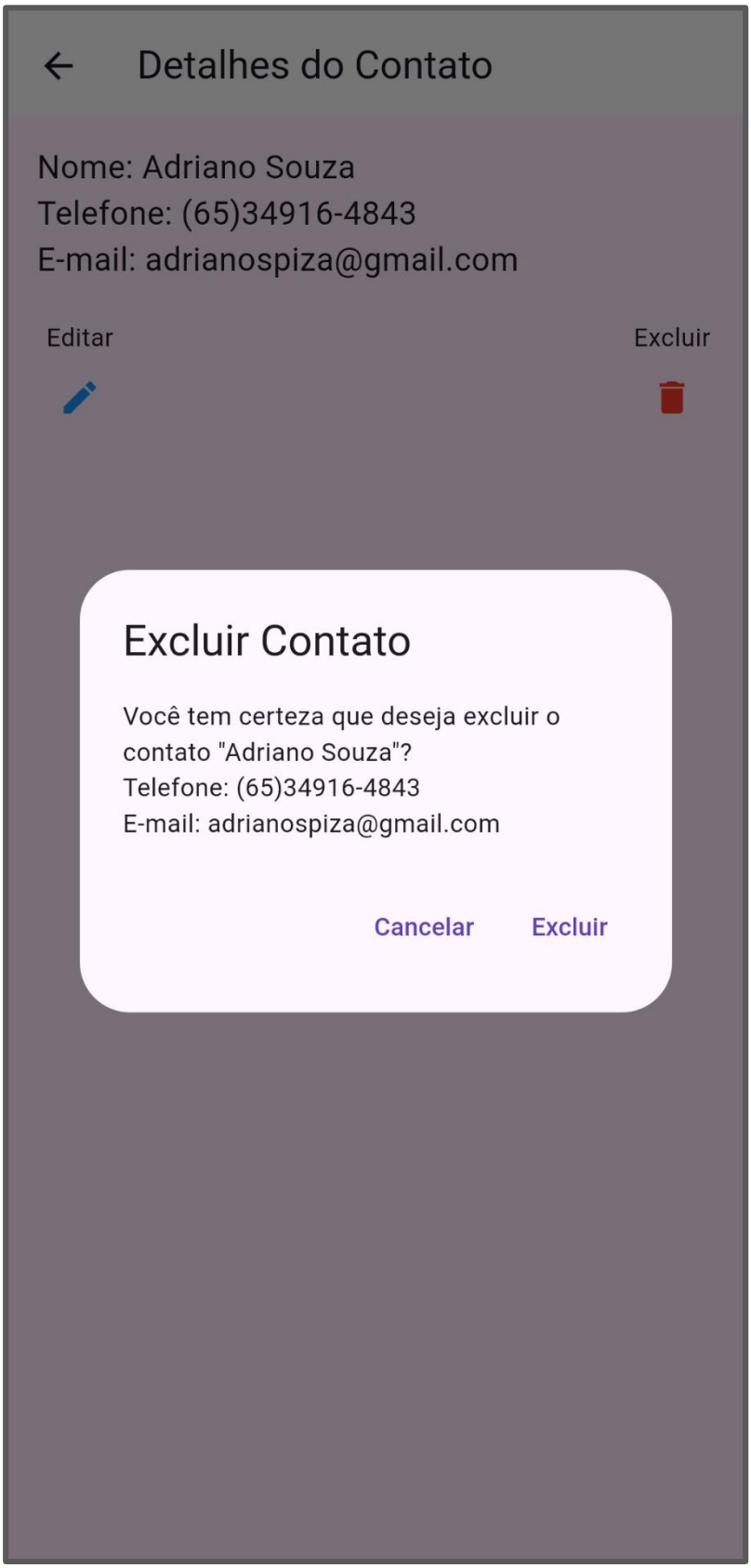
3. Update (Atualizar):

Quando o usuário edita um contato, os novos dados são capturados do formulário. A operação de atualização é realizada utilizando o comando SQL UPDATE, que modifica os dados do contato na tabela contacts com base no seu identificador único (id). Isso garante que apenas o registro correto seja atualizado no banco de dados.



4. Delete (Excluir):

A exclusão de um contato é realizada ao selecionar um contato da lista e confirmar a ação. O aplicativo utiliza o comando SQL DELETE para remover o contato correspondente da tabela contacts, com base no seu identificador único (id). Isso assegura que apenas o contato selecionado seja excluído do banco de dados.



6 Conclusão do Projeto

O desenvolvimento do Projeto Integrador V–A foi uma experiência enriquecedora, permitindo aplicar conhecimentos teóricos na criação de uma solução prática. O aplicativo de gerenciamento de contatos foi implementado com sucesso, utilizando as tecnologias Dart, Flutter e SQLite.

Durante o projeto, foi possível aprender sobre a importância da organização modular do código, a aplicação da metodologia Extreme Programming (XP) e a utilização de ferramentas como Figma para prototipação e Git/GitHub para controle de versão.

Além disso, enfrentar desafios técnicos, como a implementação do banco de dados SQLite e a integração com a interface do Flutter, contribuiu significativamente para o meu crescimento como desenvolvedor.

O projeto está disponível publicamente no GitHub, para que professores, colegas ou qualquer interessado possam acessá-lo. O link para o repositório é:

https://github.com/IsaiasDevv/Projeto_Integrador_V-A.git

O protótipo do aplicativo, desenvolvido no Figma, pode ser visualizado em:

https://www.figma.com/design/KrfuvZFtqo9LZ9cQohgp2o/Prot%C3%B3tipo?node-id=0-1&t=hPcbkOGrMCZUCg3C-1

Esses recursos demonstram o progresso alcançado e servem como base para futuros projetos e estudos. O aprendizado adquirido será fundamental para o desenvolvimento de soluções mais complexas no futuro.

Pode ser também encontrar o link do Google Drive que contém todos os arquivos juntamente um junto com o outro

https://drive.google.com/drive/folders/17maEOzMvUSLImEPCQvarv03mixRblp4l?usp=sharing

---

Link do Flutter: https://flutter.dev/

Link do Dart: https://dart.dev/

Link do SQLLite: https://www.sqlite.org/

Link do Visual Studio Code: https://code.visualstudio.com/

Link do Figma: https://www.figma.com/pt-br/

Link do Git: https://git-scm.com/

Link do GitHub: https://github.com/