

Documentação Projeto Pet Adopt.

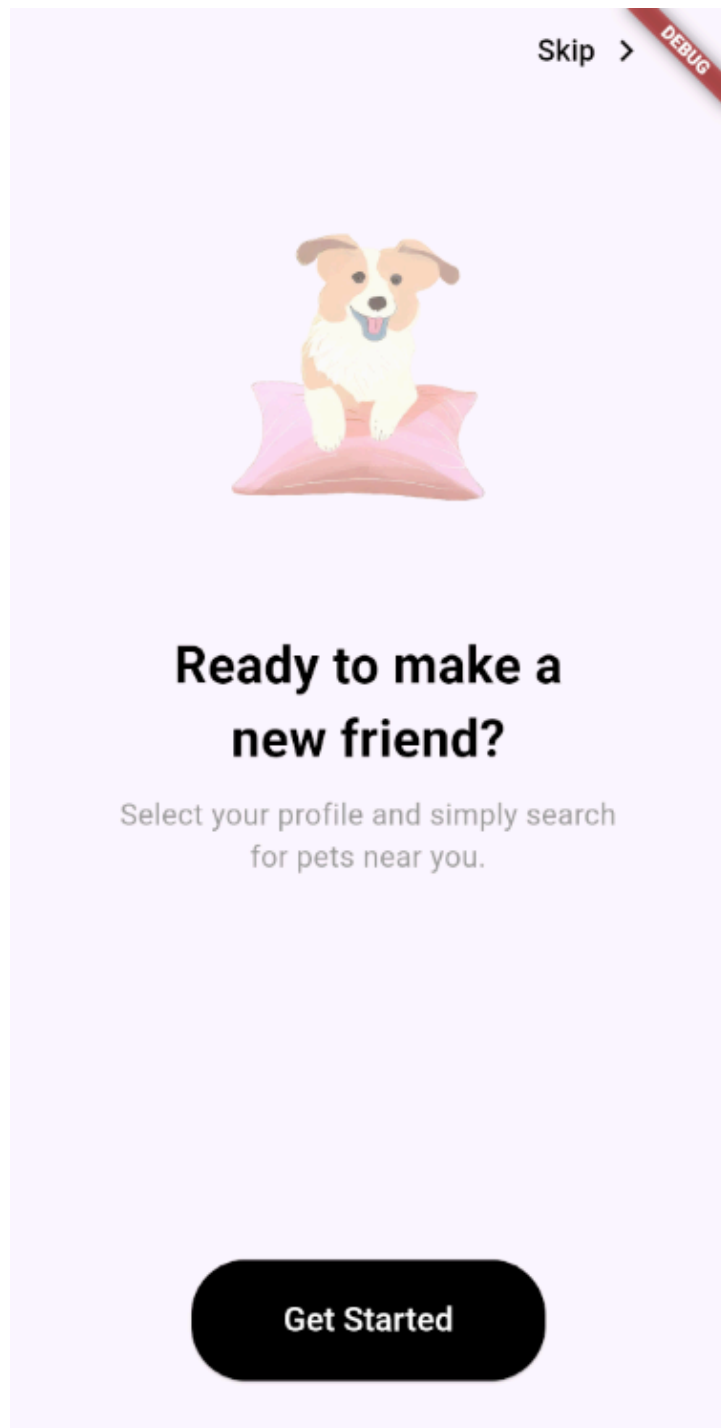


Qual o objetivo do projeto?

A aplicação **Pet Adopt** tem como objetivo principal facilitar o processo de adoção de pets, oferecendo uma plataforma intuitiva onde os usuários podem visualizar pets disponíveis para adoção, cadastrar novos pets e gerenciar seus próprios registros. O projeto promove a conexão entre adotantes e pets, incentivando a adoção responsável e segura.

Navegação entre rotas:

Home:



A HomeScreen serve como tela inicial do aplicativo, apresentando o propósito de adoção de animais. Ela exibe uma imagem temática de um cachorro e textos motivacionais como "Ready to make a new friend?". O objetivo é engajar o usuário, fornecendo botões para login ("Skip") e cadastro ("Get Started"). O usuário é incentivado a criar um perfil para buscar pets próximos.


Botão "Skip": Posicionado no canto superior direito, permite que o usuário pule o cadastro e acesse diretamente a tela de login. A navegação é realizada pelo Navigator com a rota para LoginScreen:

```
child: ElevatedButton(  
  onPressed: () {  
    Navigator.of(context).push(  
      MaterialPageRoute(builder: (context) => const LoginScreen()));  
  },  
);
```

Botão "Get Started": Localizado na parte inferior, conduz o usuário à tela de cadastro com uma navegação semelhante ao botão "Skip":

```
ElevatedButton(  
  onPressed: () {  
    Navigator.of(context).push(MaterialPageRoute(  
      builder: (context) => const CadastroScreen()));  
  },  
);
```

Cadastro:



Cadastro de usuário

< Voltar

Digite seu nome

Digite seu email

Digite seu telefone

Digite sua senha

Confirme sua senha

Cadastrar

A tela de cadastro permite ao usuário inserir informações como nome, e-mail, telefone e senha. Ela possui campos de entrada para cada dado e um botão para enviar os dados e registrar o usuário. O botão de "Cadastrar" está localizado na parte inferior da tela. A tela também conta com um botão de "Voltar" para retornar à tela anterior, proporcionando a navegação entre as páginas. Essa tela visa coletar informações essenciais para criar um novo usuário na plataforma PetAdopt.

O arquivo UserController desempenha um papel crucial na funcionalidade de cadastro de usuário. Ele é responsável por enviar os dados preenchidos na tela de cadastro (como nome, email, telefone e senha) para a API, utilizando uma requisição HTTP POST. A função registerUser pega o modelo UserModel, converte em JSON e o envia para a URL de cadastro da API. Com base na resposta da API, ela retorna uma

mensagem de sucesso ou erro, que é exibida ao usuário, permitindo que o processo de cadastro seja completado corretamente.

```
import '../model/user_model.dart';
import 'dart:convert';
import 'package:http/http.dart' as http;

class UserController {
  Future<String> registerUser(UserModel user) async {
    final url = Uri.parse('https://pet-adopt-dq32j.ondigitalocean.app/user/register');

    try {
      final userJson = jsonEncode(user.toJson());

      print('Enviando JSON: $userJson');

      final response = await http.post(
        url,
        headers: {'Content-Type': 'application/json'},
        body: userJson,
      );

      print('Resposta da API: ${response.body}');

      if (response.statusCode == 200) {
        return 'Usuário cadastrado com sucesso!';
      } else {
        return 'Falha ao registrar usuário: ${response.body}';
      }
    } catch (e) {
      return 'Erro ao registrar: $e';
    }
  }
}
```

Login:



The image shows a mobile application screen for user login. At the top, the title "Login de usuário" is centered. In the top right corner, there is a red diagonal banner with the word "DEBUG" in white. Below the title, on the left, is a back arrow icon followed by the text "Voltar". The main content area contains two text input fields: the first is labeled "Digite seu email" and the second is labeled "Digite sua senha". Below these fields are two black buttons with white text: "Entrar" on the left and "Não possuo conta" on the right.

A tela de login permite que o usuário insira seu e-mail e senha para realizar a autenticação no sistema. O fluxo é o seguinte: quando o usuário clica no botão "Entrar", os dados de login são validados e, se corretos, o token de autenticação é armazenado usando o `SharedPreferences` para permitir o acesso às telas protegidas. Caso o login falhe, uma mensagem de erro é exibida. Caso o usuário não tenha conta, é possível ser redirecionado para a tela de cadastro.

O `AuthController` gerencia o processo de login e logout de um usuário. Na função `loginUser`, ele envia o e-mail e a senha para a API de login. Se a resposta for positiva (status 200), o token de autenticação é extraído da resposta e armazenado localmente usando `SharedPreferences`. Se a autenticação falhar, a função retorna null. No método `logoutUser`, o token é removido do armazenamento local, efetivando o logout do

usuário. Este controlador é utilizado pela tela de login para gerenciar o fluxo de autenticação.

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';

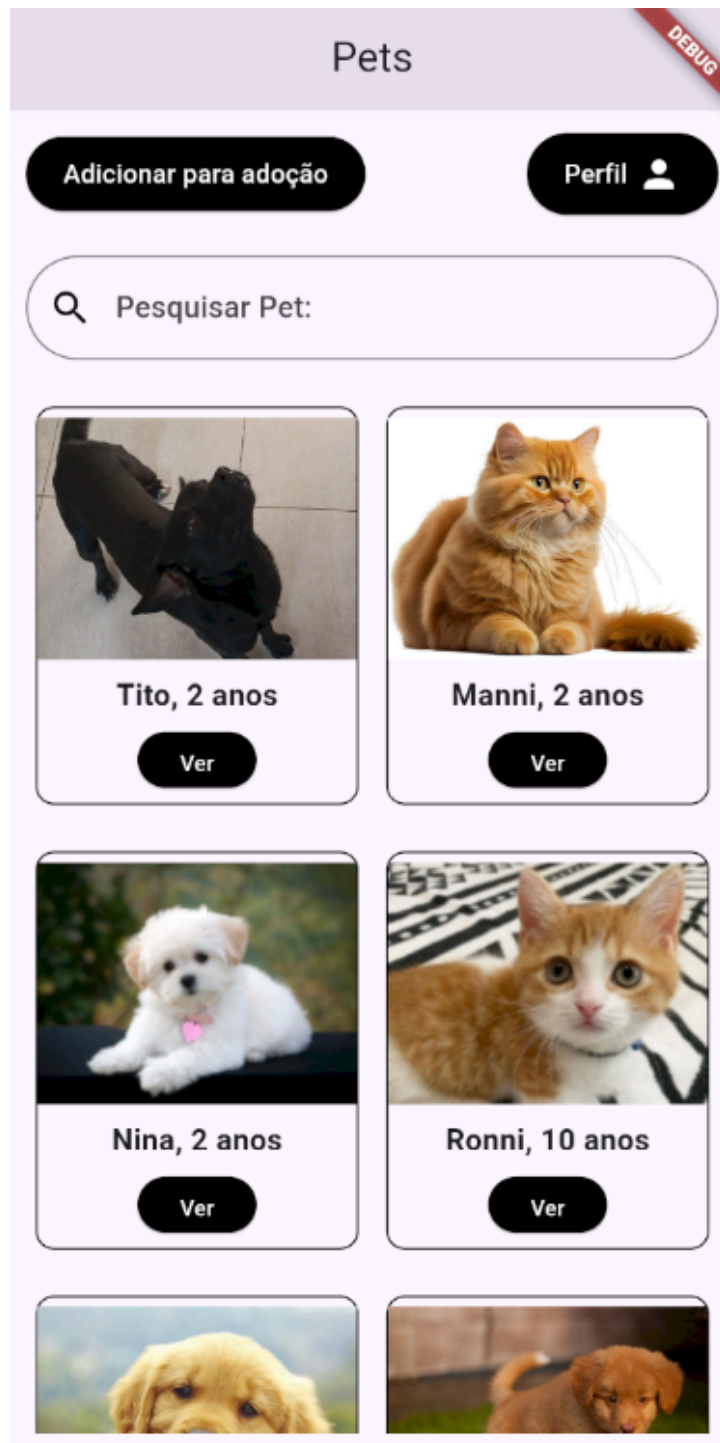
class AuthController {
  Future<String?> loginUser(String email, String password) async {
    final url = Uri.parse('https://pet-adopt-dq32j.ondigitalocean.app/user/login');

    try {
      final response = await http.post(
        url,
        headers: {'Content-Type': 'application/json'},
        body: jsonEncode({'email': email, 'password': password}),
      );

      if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        String token = data['token'];
        SharedPreferences prefs = await SharedPreferences.getInstance();
        await prefs.setString('auth_token', token);
        return token;
      } else {
        return null;
      }
    } catch (e) {
      print('Erro ao fazer login: $e');
      return null;
    }
  }

  Future<void> logoutUser() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.remove('auth_token');
  }
}
```

PetsScreen:



A tela PetsScreen exibe uma lista de pets disponíveis para adoção, com opções para adicionar novos pets ou acessar o perfil do usuário. A tela inclui uma barra de pesquisa para procurar pets e exibe os resultados em uma grade de cartões (PetCardScreen). O FutureBuilder é usado para buscar os dados dos pets via API, e enquanto os dados estão sendo carregados, um indicador de progresso é mostrado. Caso haja um erro ou nenhum pet seja encontrado, mensagens apropriadas são exibidas.

O código da tela PetCardScreen está bem estruturado para exibir informações sobre um pet, como sua imagem, nome e idade, dentro de um cartão. Ao clicar no botão

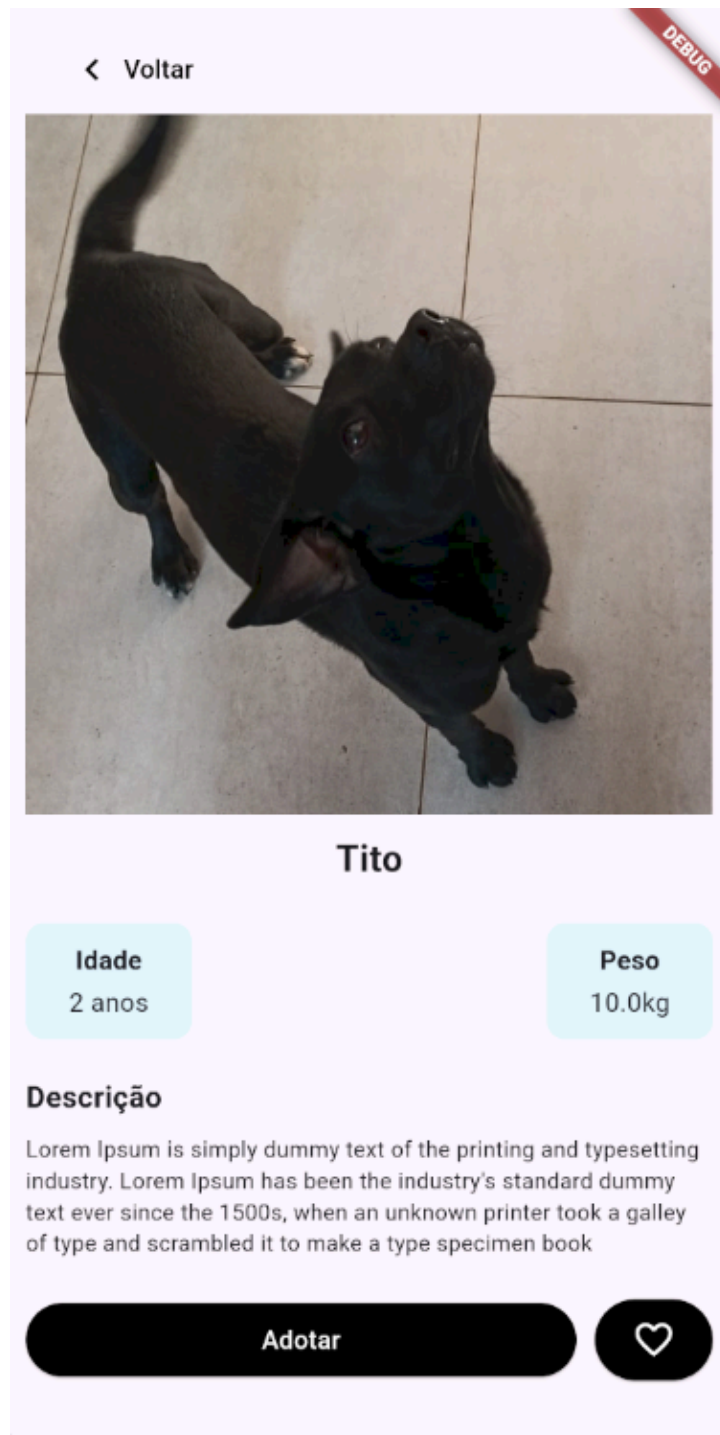
"Ver", o usuário é redirecionado para a tela PetDescScreen, onde as informações detalhadas do pet são mostradas.

```
@override
PetsScreenState createState() => _PetsScreenState();
}

class _PetsScreenState extends State<PetsScreen> {
  late Future<List<PetModel>> pets;

  @override
  void initState() {
    super.initState();
    pets = PetController().fetchPets();
  }
}
```

Desc:



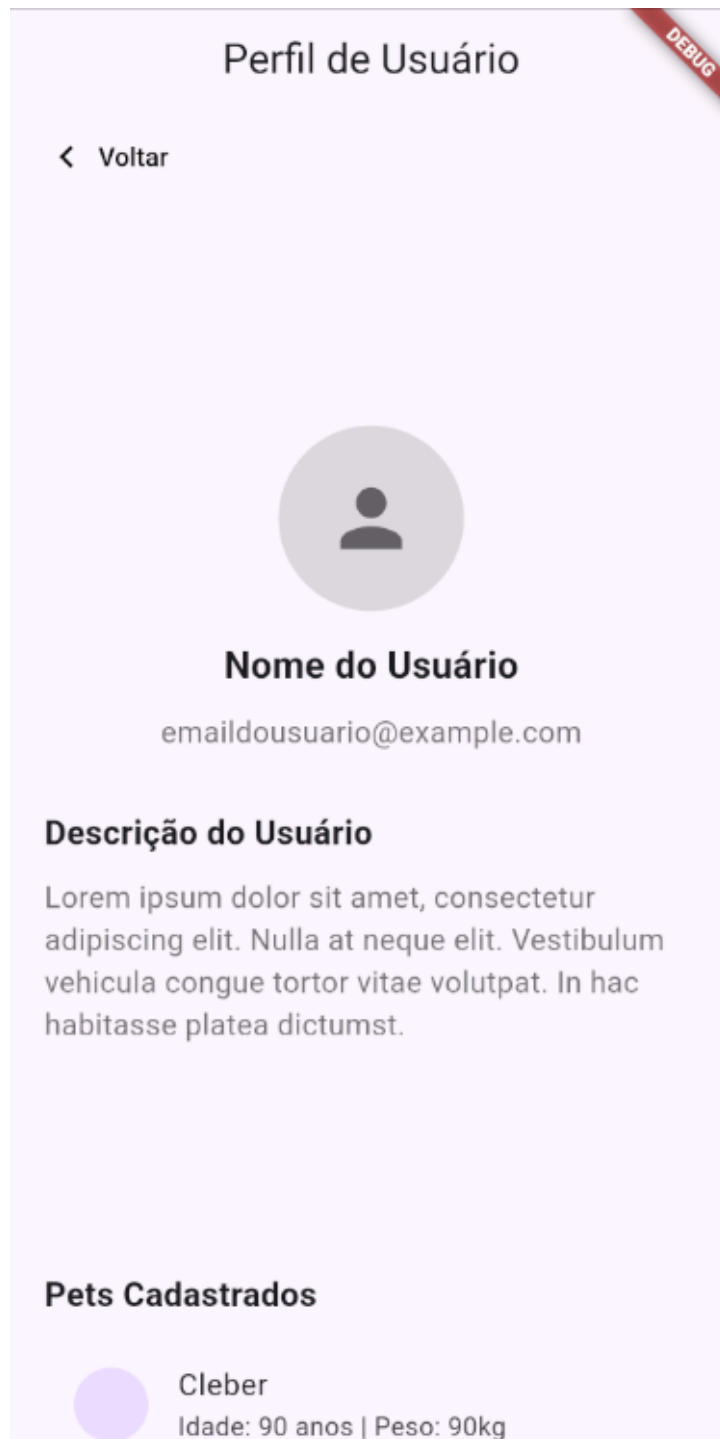
A tela PetDescScreen exibe detalhes sobre um pet, incluindo sua imagem, nome, idade, peso e uma descrição. Ela possui um botão para voltar à tela anterior e exibe as informações do pet em caixas específicas, com um layout organizado. Além disso, oferece botões para adotar ou favoritar o pet, cujas funcionalidades podem ser implementadas futuramente. A estrutura visa proporcionar uma navegação intuitiva, permitindo que o usuário acesse facilmente as informações detalhadas do pet e interaja com as opções de adoção ou favoritos.

```
final PetModel dog;
```

```
const PetDescScreen({super.key, required this.dog});
```

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceBetween,  
  children: [  
    _buildInfoBox("Idade", "${dog.age} anos"),  
    _buildInfoBox("Peso", "${dog.weight}kg"),  
  ],
```

Profile:



A tela ProfileScreen exibe informações do perfil do usuário, incluindo nome, email e descrição, além de listar os pets cadastrados por ele. Ela busca os dados dos pets de uma API protegida com autenticação via token armazenado no dispositivo, utilizando SharedPreferences. Com uma interface organizada, o perfil mostra uma foto de avatar, descrição do usuário, e uma lista dos pets cadastrados com detalhes como nome, idade, e peso. A funcionalidade é implementada com FutureBuilder para tratar dados assíncronos da API e possíveis estados como erro ou ausência de pets.

```

Future<String?> _getToken() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  return prefs.getString('auth_token');
}

Future<List<dynamic>> fetchUserPets() async {
  final token = await _getToken();
  if (token == null) {
    throw Exception('Token não encontrado. Faça login novamente.');
```

```

    final response = await http.get(
      Uri.parse('https://pet-adopt-dq32j.ondigitalocean.app/pet/mypets'),
      headers: {'Authorization': 'Bearer $token'},
    );
```

```

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      return data['pets'] as List<dynamic>;
    } else {
      throw Exception('Erro ao carregar os pets.');
```

```


  }
}
```

Conclusão:

O projeto **PetAdopt** alcança seu objetivo ao conectar tecnologia e funcionalidade, fornecendo uma aplicação completa para adoção de pets. Utilizando o Flutter e seguindo o padrão MVC, foram implementadas telas intuitivas e integradas a uma API, oferecendo recursos como cadastro de usuários, login, visualização de pets disponíveis e gerenciamento dos próprios animais. O uso de boas práticas como autenticação com SharedPreferences e tratamento assíncrono com FutureBuilder garantiu robustez e fluidez. O resultado é uma aplicação funcional, esteticamente agradável e alinhada aos propósitos sociais do projeto.

Repositório:

O código fonte está disponível no GitHub:

 [GitHub - lucassdolv/PetAdopt](https://github.com/lucassdolv/PetAdopt)