Seção 11: Autenticação com Supabase - Guia Completo

Desenvolvimento de Aplicativos Móveis - Flutter

Introdução

Nas seções anteriores, exploramos conceitos teóricos de autenticação, arquitetura de código, gerenciamento de estado com Riverpod, integração com APIs e métodos alternativos de autenticação. Agora, chegamos à **seção mais importante deste material**: a implementação prática completa usando o **Supabase**.

O Supabase é uma plataforma Backend-as-a-Service (BaaS) de código aberto que oferece autenticação pronta, banco de dados PostgreSQL, armazenamento de arquivos, funções serverless e muito mais. Para autenticação mobile, o Supabase é uma escolha excelente porque:

- **Autenticação completa out-of-the-box**: Email/senha, OAuth social, magic links, autenticação telefônica
- **SDKs oficiais bem mantidos**: Incluindo suporte completo para Flutter/Dart
- **Segurança robusta**: JWT tokens, Row Level Security (RLS), políticas de acesso
- **Gratuito para começar**: Tier gratuito generoso para desenvolvimento e projetos pequenos
- **Open source**: Você pode até self-host se precisar de controle total

Nesta seção, você aprenderá **passo a passo** como:

1. Criar e configurar um projeto Supabase
2. Integrar o Supabase SDK no Flutter
3. Implementar todos os métodos de autenticação
4. Integrar com o AuthNotifier do Riverpod (da Seção 7)
5. Configurar deep linking para confirmações por email
6. Implementar login social (Google, GitHub)
7. Implementar magic links (autenticação sem senha)
8. Resolver problemas comuns (troubleshooting)

⚠️ **Importante**: Esta seção contém muito código. Leia com atenção os comentários e teste cada etapa no seu próprio projeto para consolidar o aprendizado.

Parte 1: Criando e Configurando o Projeto Supabase

1.1. Criando uma Conta e Projeto

Antes de escrever qualquer código, precisamos configurar o backend no Supabase:

Passo 1: Criar conta

1. Acesse https://supabase.com
2. Clique em "Start your project" ou "Sign Up"
3. Você pode criar conta com GitHub, GitLab, Google ou email

Passo 2: Criar um novo projeto

1. No dashboard, clique em "New Project"
2. Escolha sua organização (ou crie uma nova)
3. Preencha os campos:
   - **Project Name**: Ex: "MeuApp Auth" (nome amigável para você)
   - **Database Password**: Crie uma senha FORTE (você precisará dela apenas para acessar o banco diretamente)
   - **Region**: Escolha a região mais próxima dos seus usuários (ex: South America para Brasil)
   - **Pricing Plan**: "Free" é suficiente para começar
4. Clique em "Create new project"
5. Aguarde 1-2 minutos enquanto o Supabase provisiona sua infraestrutura

Passo 3: Localizar suas credenciais

Após o projeto ser criado, você precisa de duas informações essenciais:

1. No dashboard do seu projeto, vá em **Settings** (ícone de engrenagem) → **API**
2. Localize e copie:
   - **Project URL**: Ex: `https://xyzabcdefgh.supabase.co`
   - **anon/public key**: Uma chave longa começando com `eyJ...`

⚠️ **Segurança**: A `anon key` é segura para usar em aplicativos mobile. Ela permite acesso público apenas às operações que você configurar no RLS (Row Level Security). Nunca compartilhe a `service_role key` em código cliente!

```dart
```

```
// ❌ NUNCA faça isso no código do app
const serviceRoleKey = 'eyJh...'; // Esta chave tem acesso total ao banco!

// ✅ Use apenas a anon key no cliente
const anonKey = 'eyJh...'; // Esta chave respeita as políticas RLS
```

1.2. Configurando Autenticação no Dashboard

Por padrão, o Supabase já vem com autenticação por email/senha habilitada. Mas vamos revisar as configurações:

1. Authentication Settings

1. No menu lateral, vá em **Authentication → Settings**
2. Em **Email Auth**, verifique:
   - ✅ "Enable Email Signups" (permite cadastro)
   - ✅ "Enable Email Confirmations" (requer confirmação por email)
   - ⚠️ Para desenvolvimento, você pode desabilitar "Email Confirmations" temporariamente

2. Email Templates

1. Ainda em **Authentication → Email Templates**
2. Aqui você encontra templates para:
   - **Confirm Signup**: Email de confirmação de cadastro
   - **Magic Link**: Email com link de acesso direto
   - **Reset Password**: Email de recuperação de senha

Você pode customizar esses templates mais tarde (veremos na Parte 7).

3. URL Configuration

1. Em **Authentication → URL Configuration**
2. Configure o **Site URL**: Ex: `https://meuapp.com` (pode deixar padrão por enquanto)
3. Configure **Redirect URLs**: URLs que o Supabase pode redirecionar após autenticação
   - Para desenvolvimento Flutter: `io.supabase.meuapp://login-callback`
   - Adicione também: `https://seu-dominio.com/auth/callback` (para web, se aplicável)

⚠️ **Importante**: O esquema de URL (`io.supabase.meuapp`) deve ser único para seu app. Vamos configurar isso no Flutter em breve.

Parte 2: Instalando e Configurando o Supabase no Flutter

2.1. Adicionando Dependências

Abra o arquivo `pubspec.yaml` do seu projeto Flutter e adicione:

```yaml
dependencies:
  flutter:
    sdk: flutter

  # Supabase
  supabase_flutter: ^2.5.0  # Versão mais recente em 2024

  # Gerenciamento de estado (se ainda não tiver)
  flutter_riverpod: ^2.5.0

  # Para deep linking (opcional, mas recomendado)
  app_links: ^6.0.0

  # Armazenamento seguro (o Supabase já inclui internamente)
  # flutter_secure_storage: ^9.0.0  # Não é mais necessário adicionar manualmente
```

Execute no terminal:

```bash
flutter pub get
```

Por que `supabase_flutter` e não `supabase`?

O pacote `supabase_flutter` é específico para Flutter e inclui:

- Integração com flutter_secure_storage (armazena tokens automaticamente)
- Deep linking helpers
- Configuração facilitada
- Suporte a hot reload

2.2. Configurando Deep Linking (Android e iOS)

Para que magic links, confirmações de email e OAuth funcionem, precisamos configurar deep linking.

Android Configuration

1. Abra `android/app/src/main/AndroidManifest.xml`

Adicione dentro da tag `<activity>` (onde está `android:name=".MainActivity"`):

```xml
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:launchMode="singleTop"
    android:theme="@style/LaunchTheme"
    android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontS
    android:hardwareAccelerated="true"
    android:windowSoftInputMode="adjustResize">

    <!-- Configuração existente de flutter -->
    <meta-data
      android:name="io.flutter.embedding.android.NormalTheme"
      android:resource="@style/NormalTheme" />

    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <!-- 🔷 ADICIONE ESTE INTENT FILTER para deep linking -->
    <intent-filter android:autoVerify="true">
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />

      <!-- Substitua 'io.supabase.meuapp' pelo seu próprio esquema -->
      <data
        android:scheme="io.supabase.meuapp"
        android:host="login-callback" />
    </intent-filter>

</activity>
```

⚠️ **Importante**: Troque `io.supabase.meuapp` por um esquema único para seu app. Use seu domínio invertido: `com.seudominio.seuapp`

iOS Configuration

1. Abra `ios/Runner/Info.plist`

Adicione antes da última tag `</dict>`:

```xml
<!-- Deep linking configuration -->
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleTypeRole</key>
    <string>Editor</string>
    <key>CFBundleURLSchemes</key>
    <array>
      <!-- Substitua 'io.supabase.meuapp' pelo seu esquema -->
      <string>io.supabase.meuapp</string>
    </array>
  </dict>
</array>
```

2. Também adicione (para links universais - opcional mas recomendado):

```xml
<key>FlutterDeepLinkingEnabled</key>
<true/>
```

2.3. Inicializando o Supabase no Flutter

Agora vamos inicializar o Supabase quando o app inicia.

Abra ou crie `lib/main.dart`:

```dart
```

```dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

// Suas credenciais do Supabase
// ⚠️ IMPORTANTE: Em produção, considere usar variáveis de ambiente
const supabaseUrl = 'https://xyzabcdefgh.supabase.co'; // Sua Project URL
const supabaseAnonKey = 'eyJh...'; // Sua anon key

void main() async {
  // Garante que os widgets do Flutter estão inicializados
  WidgetsFlutterBinding.ensureInitialized();

  // Inicializa o Supabase ANTES de rodar o app
  await Supabase.initialize(
    url: supabaseUrl,
    anonKey: supabaseAnonKey,

    // Configurações opcionais mas recomendadas:
    authOptions: FlutterAuthClientOptions(
      // Define o esquema de deep link (deve ser o mesmo do AndroidManifest/Info.plist)
      authFlowType: AuthFlowType.pkce, // Mais seguro que 'implicit'

      // Para deep linking funcionar corretamente
      // Não é mais necessário passar authCallbackUrlHostname separadamente
      // O pacote detecta automaticamente do esquema configurado
    ),

    // Configurações de armazenamento (usa flutter_secure_storage internamente)
    // O Supabase já cuida disso automaticamente - tokens são armazenados de forma segura
  );

  runApp(
    // ProviderScope do Riverpod para gerenciamento de estado
    const ProviderScope(
      child: MyApp(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
      title: 'Auth com Supabase',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const AuthWrapper(), // Vamos criar isso em breve
    );
  }
}

// Classe helper para acessar o Supabase client em qualquer lugar do app
final supabase = Supabase.instance.client;
```

🔍 **Explicando as configurações:**

1. **`authFlowType: AuthFlowType.pkce`**:
   - PKCE (Proof Key for Code Exchange) é mais seguro que o fluxo implícito
   - Recomendado para apps mobile
   - Protege contra ataques de interceptação
2. **Por que inicializar no `main()`?**
   - O Supabase precisa estar pronto antes de qualquer tela ser carregada
   - Ele restaura automaticamente a sessão do usuário se houver token válido
   - Evita erros de "Supabase não inicializado"
3. **Variável global `supabase`**:
   - `Supabase.instance.client` dá acesso ao cliente Supabase
   - Criar uma variável global facilita o uso em todo o código
   - Alternativa: criar um provider Riverpod (veremos adiante)

2.4. Testando a Inicialização

Vamos criar uma tela simples para testar se tudo está funcionando:

dart

```dart
// lib/screens/home_screen.dart
import 'package:flutter/material.dart';
import '../main.dart'; // Para acessar a variável 'supabase'

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'Supabase inicializado com sucesso!',
              style: Theme.of(context).textTheme.titleLarge,
            ),
            const SizedBox(height: 20),
            // Mostra o estado atual da autenticação
            FutureBuilder(
              future: supabase.auth.currentSession,
              builder: (context, snapshot) {
                if (snapshot.connectionState == ConnectionState.waiting) {
                  return const CircularProgressIndicator();
                }

                final session = snapshot.data;
                if (session != null) {
                  return Text('Usuário logado: ${session.user.email}');
                } else {
                  return const Text('Nenhum usuário logado');
                }
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

Execute o app. Se não houver erros, significa que o Supabase foi inicializado corretamente!

Parte 3: Implementações Práticas - Autenticação Tradicional

Agora vamos implementar cada método de autenticação. Começaremos com o mais básico: email e senha.

3.1. Cadastro de Usuário (Sign Up)

O método `signUp` cria uma nova conta no Supabase Auth.

Criando um AuthService:

dart

```dart
// lib/services/supabase_auth_service.dart
import 'package:supabase_flutter/supabase_flutter.dart';
import '../main.dart';

class SupabaseAuthService {
  // Referência ao cliente Supabase
  final SupabaseClient _supabase = supabase;

  /// Cadastra um novo usuário com email e senha
  ///
  /// [email]: Email do usuário
  /// [password]: Senha (mínimo 6 caracteres por padrão no Supabase)
  /// [data]: Metadados opcionais do usuário (nome, avatar, etc)
  ///
  /// Retorna a AuthResponse com o usuário e sessão
  /// Lança GoTrueException se houver erro
  Future<AuthResponse> signUp({
    required String email,
    required String password,
    Map<String, dynamic>? data, // Metadados do usuário
  }) async {
    try {
      final response = await _supabase.auth.signUp(
        email: email,
        password: password,
        data: data, // Ex: {'name': 'João Silva', 'avatar_url': '...'}

        // Opções adicionais:
        emailRedirectTo: 'io.supabase.meuapp://login-callback',
      );

      // Verificar se requer confirmação de email
      if (response.user != null && !response.user!.emailConfirmedAt) {
        // Usuário criado mas precisa confirmar email
        // Supabase envia automaticamente o email de confirmação
        print('✉️ Email de confirmação enviado para ${response.user!.email}');
      }

      return response;

    } on AuthException catch (e) {
      // Erros específicos do Supabase Auth
      throw _handleAuthException(e);

    } catch (e) {
      // Outros erros (rede, etc)
```

```
      throw Exception('Erro ao cadastrar: $e');
    }
  }

  /// Converte AuthException em mensagens amigáveis
  String _handleAuthException(AuthException exception) {
    switch (exception.statusCode) {
      case '400':
        if (exception.message.contains('already registered')) {
          return 'Este email já está cadastrado';
        }
        return 'Dados inválidos. Verifique email e senha.';

      case '422':
        if (exception.message.contains('email')) {
          return 'Email inválido';
        }
        if (exception.message.contains('password')) {
          return 'Senha muito fraca. Use no mínimo 6 caracteres.';
        }
        return exception.message;

      case '429':
        return 'Muitas tentativas. Aguarde alguns minutos.';

      default:
        return exception.message;
    }
  }
}
```

Usando o signUp em uma tela:

dart

```dart
// lib/screens/signup_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';

class SignUpScreen extends ConsumerStatefulWidget {
  const SignUpScreen({super.key});

  @override
  ConsumerState<SignUpScreen> createState() => _SignUpScreenState();
}

class _SignUpScreenState extends ConsumerState<SignUpScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _nameController = TextEditingController();

  bool _isLoading = false;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    _nameController.dispose();
    super.dispose();
  }

  Future<void> _handleSignUp() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => _isLoading = true);

    try {
      final authService = SupabaseAuthService();

      final response = await authService.signUp(
        email: _emailController.text.trim(),
        password: _passwordController.text,
        data: {
          'name': _nameController.text.trim(), // Metadado customizado
        },
      );

      if (!mounted) return;
```

```
    // Verificar se precisa confirmar email
    if (response.user != null && response.user!.emailConfirmedAt == null) {
      // Mostrar mensagem pedindo para verificar email
      _showSuccessDialog(
        'Cadastro realizado!',
        'Enviamos um email de confirmação para ${response.user!.email}. '
        'Clique no link para ativar sua conta.',
      );
    } else {
      // Login automático (se confirmação de email estiver desabilitada)
      Navigator.of(context).pushReplacementNamed('/home');
    }

  } catch (e) {
    if (!mounted) return;

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(e.toString()),
        backgroundColor: Colors.red,
      ),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

void _showSuccessDialog(String title, String message) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: Text(title),
      content: Text(message),
      actions: [
        TextButton(
          onPressed: () {
            Navigator.of(context).pop();
            Navigator.of(context).pushReplacementNamed('/login');
          },
          child: const Text('OK'),
        ),
      ],
    ),
  );
}
```

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Criar Conta')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextFormField(
              controller: _nameController,
              decoration: const InputDecoration(
                labelText: 'Nome',
                border: OutlineInputBorder(),
              ),
              validator: (value) {
                if (value == null || value.trim().isEmpty) {
                  return 'Digite seu nome';
                }
                return null;
              },
            ),
            const SizedBox(height: 16),

            TextFormField(
              controller: _emailController,
              decoration: const InputDecoration(
                labelText: 'Email',
                border: OutlineInputBorder(),
              ),
              keyboardType: TextInputType.emailAddress,
              validator: (value) {
                if (value == null || !value.contains('@')) {
                  return 'Digite um email válido';
                }
                return null;
              },
            ),
            const SizedBox(height: 16),

            TextFormField(
              controller: _passwordController,
              decoration: const InputDecoration(
                labelText: 'Senha',
```

```
            border: OutlineInputBorder(),
          ),
          obscureText: true,
          validator: (value) {
            if (value == null || value.length < 6) {
              return 'Senha deve ter no mínimo 6 caracteres';
            }
            return null;
          },
        ),
        const SizedBox(height: 24),

        SizedBox(
          width: double.infinity,
          height: 48,
          child: ElevatedButton(
            onPressed: _isLoading ? null : _handleSignUp,
            child: _isLoading
                ? const CircularProgressIndicator()
                : const Text('Criar Conta'),
          ),
        ),
      ],
    ),
  ),
),
);
}
}
```

3.2. Login Tradicional (Sign In with Password)

O método `signInWithPassword` autentica um usuário existente.

Adicione ao AuthService:

dart

```dart
// lib/services/supabase_auth_service.dart

/// Faz login com email e senha
///
/// Retorna a AuthResponse com o usuário e sessão
/// Lança Exception se credenciais inválidas
Future<AuthResponse> signIn({
  required String email,
  required String password,
}) async {
  try {
    final response = await _supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );

    // Verificar se o email foi confirmado (se confirmação estiver habilitada)
    if (response.user != null && response.user!.emailConfirmedAt == null) {
      // Email não confirmado
      throw Exception(
        'Email não confirmado. Verifique sua caixa de entrada.',
      );
    }

    return response;

  } on AuthException catch (e) {
    throw _handleAuthException(e);
  } catch (e) {
    throw Exception('Erro ao fazer login: $e');
  }
}
```

Atualizar o tratamento de erros:

```dart
```

```
String _handleAuthException(AuthException exception) {
  switch (exception.statusCode) {
    case '400':
      if (exception.message.contains('already registered')) {
        return 'Este email já está cadastrado';
      }
      if (exception.message.contains('Invalid login credentials')) {
        return 'Email ou senha incorretos';
      }
      return 'Dados inválidos. Verifique email e senha.';

    case '422':
      if (exception.message.contains('email')) {
        return 'Email inválido';
      }
      if (exception.message.contains('password')) {
        return 'Senha muito fraca. Use no mínimo 6 caracteres.';
      }
      return exception.message;

    case '429':
      return 'Muitas tentativas. Aguarde alguns minutos.';

    default:
      return exception.message;
  }
}
```

Tela de Login:

dart

```dart
// lib/screens/login_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';

class LoginScreen extends ConsumerStatefulWidget {
  const LoginScreen({super.key});

  @override
  ConsumerState<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends ConsumerState<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();

  bool _isLoading = false;
  bool _obscurePassword = true;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  Future<void> _handleLogin() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => _isLoading = true);

    try {
      final authService = SupabaseAuthService();

      final response = await authService.signIn(
        email: _emailController.text.trim(),
        password: _passwordController.text,
      );

      if (!mounted) return;

      // Login bem-sucedido - navegar para home
      Navigator.of(context).pushReplacementNamed('/home');

    } catch (e) {
```

```
    if (!mounted) return;

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(e.toString()),
        backgroundColor: Colors.red,
      ),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Login')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextFormField(
              controller: _emailController,
              decoration: const InputDecoration(
                labelText: 'Email',
                border: OutlineInputBorder(),
                prefixIcon: Icon(Icons.email),
              ),
              keyboardType: TextInputType.emailAddress,
              validator: (value) {
                if (value == null || !value.contains('@')) {
                  return 'Digite um email válido';
                }
                return null;
              },
            ),
            const SizedBox(height: 16),

            TextFormField(
              controller: _passwordController,
              decoration: InputDecoration(
                labelText: 'Senha',
```

```
    border: const OutlineInputBorder(),
    prefixIcon: const Icon(Icons.lock),
    suffixIcon: IconButton(
     icon: Icon(
      _obscurePassword ? Icons.visibility : Icons.visibility_off,
     ),
     onPressed: () {
      setState(() => _obscurePassword = !_obscurePassword);
     },
    ),
   ),
   obscureText: _obscurePassword,
   validator: (value) {
    if (value == null || value.isEmpty) {
     return 'Digite sua senha';
    }
    return null;
   },
  ),
  const SizedBox(height: 8),

  Align(
   alignment: Alignment.centerRight,
   child: TextButton(
    onPressed: () {
     Navigator.of(context).pushNamed('/forgot-password');
    },
    child: const Text('Esqueci minha senha'),
   ),
  ),
  const SizedBox(height: 24),

  SizedBox(
   width: double.infinity,
   height: 48,
   child: ElevatedButton(
    onPressed: _isLoading ? null : _handleLogin,
    child: _isLoading
      ? const CircularProgressIndicator(color: Colors.white)
      : const Text('Entrar'),
   ),
  ),
  const SizedBox(height: 16),

  TextButton(
   onPressed: () {
    Navigator.of(context).pushNamed('/signup');
```

```
            },
            child: const Text('Não tem conta? Cadastre-se'),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

3.3. Recuperação de Senha (Password Reset)

O fluxo de recuperação de senha tem duas etapas:

1. Usuário solicita reset (recebe email com link)
2. Usuário clica no link e define nova senha

Etapa 1: Solicitar reset de senha

dart

```dart
// lib/services/supabase_auth_service.dart

/// Envia email de recuperação de senha
///
/// [email]: Email do usuário que esqueceu a senha
///
/// Supabase envia automaticamente um email com link de reset
Future<void> resetPasswordForEmail(String email) async {
 try {
  await _supabase.auth.resetPasswordForEmail(
   email,
   redirectTo: 'io.supabase.meuapp://reset-password',
  );

  // Sucesso - email enviado (Supabase não retorna erro mesmo se email não existir)
  // Isso é por segurança - para não vazar quais emails estão cadastrados

 } on AuthException catch (e) {
  throw _handleAuthException(e);
 } catch (e) {
  throw Exception('Erro ao solicitar recuperação: $e');
 }
}
```

Tela para solicitar recuperação:

dart

```dart
// lib/screens/forgot_password_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';

class ForgotPasswordScreen extends ConsumerStatefulWidget {
  const ForgotPasswordScreen({super.key});

  @override
  ConsumerState<ForgotPasswordScreen> createState() => _ForgotPasswordScreenState();
}

class _ForgotPasswordScreenState extends ConsumerState<ForgotPasswordScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();

  bool _isLoading = false;
  bool _emailSent = false;

  @override
  void dispose() {
    _emailController.dispose();
    super.dispose();
  }

  Future<void> _handleResetRequest() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => _isLoading = true);

    try {
      final authService = SupabaseAuthService();

      await authService.resetPasswordForEmail(_emailController.text.trim());

      if (!mounted) return;

      setState(() => _emailSent = true);

    } catch (e) {
      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(e.toString()),
          backgroundColor: Colors.red,
```

```
      ),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Recuperar Senha')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: _emailSent ? _buildSuccessView() : _buildFormView(),
    ),
  );
}

Widget _buildFormView() {
  return Form(
    key: _formKey,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Icon(
          Icons.lock_reset,
          size: 64,
          color: Colors.blue,
        ),
        const SizedBox(height: 24),

        Text(
          'Esqueceu sua senha?',
          style: Theme.of(context).textTheme.headlineSmall,
        ),
        const SizedBox(height: 8),

        const Text(
          'Digite seu email e enviaremos um link para redefinir sua senha.',
          textAlign: TextAlign.center,
          style: TextStyle(color: Colors.grey),
        ),
        const SizedBox(height: 32),

        TextFormField(
```

```
          controller: _emailController,
          decoration: const InputDecoration(
            labelText: 'Email',
            border: OutlineInputBorder(),
            prefixIcon: Icon(Icons.email),
          ),
          keyboardType: TextInputType.emailAddress,
          validator: (value) {
            if (value == null || !value.contains('@')) {
              return 'Digite um email válido';
            }
            return null;
          },
        ),
        const SizedBox(height: 24),

        SizedBox(
          width: double.infinity,
          height: 48,
          child: ElevatedButton(
            onPressed: _isLoading ? null : _handleResetRequest,
            child: _isLoading
                ? const CircularProgressIndicator(color: Colors.white)
                : const Text('Enviar Link de Recuperação'),
          ),
        ),
      ],
    ),
  );
}

Widget _buildSuccessView() {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      const Icon(
        Icons.mark_email_read,
        size: 64,
        color: Colors.green,
      ),
      const SizedBox(height: 24),

      Text(
        'Email Enviado!',
        style: Theme.of(context).textTheme.headlineSmall,
      ),
      const SizedBox(height: 8),
```

```
        Text(
          'Enviamos um link de recuperação para\n${_emailController.text}',
          textAlign: TextAlign.center,
          style: const TextStyle(color: Colors.grey),
        ),
        const SizedBox(height: 16),

        const Text(
          'Verifique sua caixa de entrada e spam.',
          textAlign: TextAlign.center,
          style: TextStyle(fontSize: 12, color: Colors.grey),
        ),
        const SizedBox(height: 32),

        ElevatedButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          child: const Text('Voltar ao Login'),
        ),
      ],
    );
  }
}
```

Etapa 2: Redefinir senha (após clicar no link do email)

Quando o usuário clica no link do email, o Supabase redireciona para seu app com um token especial. Precisamos capturar esse deep link e permitir que o usuário defina uma nova senha.

dart

```dart
// lib/services/supabase_auth_service.dart

/// Atualiza a senha do usuário (usado após recuperação)
///
/// IMPORTANTE: Este método só funciona quando há um token de recuperação ativo
/// O token vem no deep link do email de recuperação
Future<void> updatePassword(String newPassword) async {
  try {
    final response = await _supabase.auth.updateUser(
      UserAttributes(password: newPassword),
    );

    if (response.user == null) {
      throw Exception('Sessão inválida. Solicite um novo link de recuperação.');
    }

  } on AuthException catch (e) {
    throw _handleAuthException(e);
  } catch (e) {
    throw Exception('Erro ao atualizar senha: $e');
  }
}
```

Tela para redefinir senha:

dart

```dart
// lib/screens/reset_password_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';
import '../main.dart';

class ResetPasswordScreen extends ConsumerStatefulWidget {
  const ResetPasswordScreen({super.key});

  @override
  ConsumerState<ResetPasswordScreen> createState() => _ResetPasswordScreenState();
}

class _ResetPasswordScreenState extends ConsumerState<ResetPasswordScreen> {
  final _formKey = GlobalKey<FormState>();
  final _passwordController = TextEditingController();
  final _confirmPasswordController = TextEditingController();

  bool _isLoading = false;
  bool _obscurePassword = true;
  bool _obscureConfirmPassword = true;

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

  @override
  void dispose() {
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

  void _checkSession() {
    // Verificar se há uma sessão válida (vindo do link de recuperação)
    final session = supabase.auth.currentSession;
    if (session == null) {
      // Não há sessão ativa - usuário não veio do link válido
      WidgetsBinding.instance.addPostFrameCallback((_) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('Link inválido ou expirado. Solicite um novo.'),
            backgroundColor: Colors.red,
          ),
```

```
      );
      Navigator.of(context).pushReplacementNamed('/forgot-password');
    });
  }
}

Future<void> _handleResetPassword() async {
  if (!_formKey.currentState!.validate()) return;

  setState(() => _isLoading = true);

  try {
    final authService = SupabaseAuthService();

    await authService.updatePassword(_passwordController.text);

    if (!mounted) return;

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Senha alterada com sucesso!'),
        backgroundColor: Colors.green,
      ),
    );

    // Redirecionar para login
    Navigator.of(context).pushReplacementNamed('/login');

  } catch (e) {
    if (!mounted) return;

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(e.toString()),
        backgroundColor: Colors.red,
      ),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
```

```
appBar: AppBar(title: const Text('Nova Senha')),
body: Padding(
 padding: const EdgeInsets.all(16.0),
 child: Form(
  key: _formKey,
  child: Column(
   mainAxisAlignment: MainAxisAlignment.center,
   children: [
    const Icon(
     Icons.lock_open,
     size: 64,
     color: Colors.blue,
    ),
    const SizedBox(height: 24),

    Text(
     'Criar Nova Senha',
     style: Theme.of(context).textTheme.headlineSmall,
    ),
    const SizedBox(height: 8),

    const Text(
     'Escolha uma senha forte e memorável.',
     textAlign: TextAlign.center,
     style: TextStyle(color: Colors.grey),
    ),
    const SizedBox(height: 32),

    TextFormField(
     controller: _passwordController,
     decoration: InputDecoration(
      labelText: 'Nova Senha',
      border: const OutlineInputBorder(),
      prefixIcon: const Icon(Icons.lock),
      suffixIcon: IconButton(
       icon: Icon(
        _obscurePassword ? Icons.visibility : Icons.visibility_off,
       ),
       onPressed: () {
        setState(() => _obscurePassword = !_obscurePassword);
       },
      ),
     ),
     obscureText: _obscurePassword,
     validator: (value) {
      if (value == null || value.length < 6) {
       return 'Senha deve ter no mínimo 6 caracteres';
```

```
            }
            return null;
          },
        ),
        const SizedBox(height: 16),

        TextFormField(
          controller: _confirmPasswordController,
          decoration: InputDecoration(
            labelText: 'Confirmar Nova Senha',
            border: const OutlineInputBorder(),
            prefixIcon: const Icon(Icons.lock_outline),
            suffixIcon: IconButton(
              icon: Icon(
                _obscureConfirmPassword
                    ? Icons.visibility
                    : Icons.visibility_off,
              ),
              onPressed: () {
                setState(() => _obscureConfirmPassword = !_obscureConfirmPassword);
              },
            ),
          ),
          obscureText: _obscureConfirmPassword,
          validator: (value) {
            if (value != _passwordController.text) {
              return 'As senhas não correspondem';
            }
            return null;
          },
        ),
        const SizedBox(height: 24),

        SizedBox(
          width: double.infinity,
          height: 48,
          child: ElevatedButton(
            onPressed: _isLoading ? null : _handleResetPassword,
            child: _isLoading
                ? const CircularProgressIndicator(color: Colors.white)
                : const Text('Redefinir Senha'),
          ),
        ),
      ],
    ),
  ),
),
```

```
    );
  }
}
```

3.4. Logout (Sign Out)

O logout é simples: limpa a sessão local e revoga o token no servidor.

dart

```dart
// lib/services/supabase_auth_service.dart

/// Faz logout do usuário atual
///
/// Remove tokens locais e revoga no servidor
Future<void> signOut() async {
  try {
    await _supabase.auth.signOut();

  } catch (e) {
    // Logout raramente falha, mas é bom tratar
    throw Exception('Erro ao fazer logout: $e');
  }
}
```

Exemplo de uso na tela de perfil:

dart

```dart
// lib/screens/profile_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';
import '../main.dart';

class ProfileScreen extends ConsumerWidget {
  const ProfileScreen({super.key});

  Future<void> _handleLogout(BuildContext context) async {
    // Mostrar confirmação
    final confirmed = await showDialog<bool>(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text('Sair'),
        content: const Text('Tem certeza que deseja sair?'),
        actions: [
          TextButton(
            onPressed: () => Navigator.of(context).pop(false),
            child: const Text('Cancelar'),
          ),
          TextButton(
            onPressed: () => Navigator.of(context).pop(true),
            child: const Text('Sair'),
          ),
        ],
      ),
    );

    if (confirmed != true) return;

    try {
      final authService = SupabaseAuthService();
      await authService.signOut();

      if (!context.mounted) return;

      // Navegar para login
      Navigator.of(context).pushReplacementNamed('/login');

    } catch (e) {
      if (!context.mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(e.toString()),
```

```
        backgroundColor: Colors.red,
      ),
    );
  }
}

@override
Widget build(BuildContext context, WidgetRef ref) {
  final user = supabase.auth.currentUser;

  return Scaffold(
    appBar: AppBar(title: const Text('Perfil')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          CircleAvatar(
            radius: 40,
            child: Text(
              user?.email?.substring(0, 1).toUpperCase() ?? '?',
              style: const TextStyle(fontSize: 32),
            ),
          ),
          const SizedBox(height: 16),

          Text(
            user?.email ?? 'Sem email',
            style: Theme.of(context).textTheme.titleLarge,
          ),
          const SizedBox(height: 8),

          Text(
            'ID: ${user?.id ?? 'N/A'}',
            style: const TextStyle(color: Colors.grey, fontSize: 12),
          ),
          const SizedBox(height: 24),

          ListTile(
            leading: const Icon(Icons.person),
            title: const Text('Editar Perfil'),
            trailing: const Icon(Icons.chevron_right),
            onTap: () {
              // Navegar para tela de edição
            },
          ),
```

```
          ListTile(
            leading: const Icon(Icons.lock),
            title: const Text('Alterar Senha'),
            trailing: const Icon(Icons.chevron_right),
            onTap: () {
              // Navegar para tela de alteração de senha
            },
          ),

          const Spacer(),

          SizedBox(
            width: double.infinity,
            height: 48,
            child: ElevatedButton.icon(
              onPressed: () => _handleLogout(context),
              icon: const Icon(Icons.logout),
              label: const Text('Sair'),
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.red,
                foregroundColor: Colors.white,
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

Parte 4: Login Social com OAuth 2.0

O Supabase suporta diversos provedores OAuth: Google, GitHub, Facebook, Twitter, Discord, e muitos outros. Vamos implementar Google e GitHub como exemplos.

4.1. Configurando Google OAuth no Supabase

Passo 1: Criar credenciais no Google Cloud Console

1. Acesse Google Cloud Console
2. Crie um novo projeto ou selecione um existente
3. Vá em **APIs & Services → Credentials**

4. Clique em **Create Credentials → OAuth 2.0 Client ID**
5. Configure consent screen se solicitado:
    - User Type: External
    - App name: Nome do seu app
    - Support email: Seu email
    - Authorized domains: (pode deixar vazio por enquanto)
6. Criar OAuth Client ID:
    - Application type: **Web application**
    - Name: "Supabase Auth"
    - Authorized redirect URIs: `https://seu-projeto.supabase.co/auth/v1/callback`
        - Substitua `seu-projeto` pelo ID do seu projeto Supabase
        - Encontre em: Supabase Dashboard → Settings → API → Project URL
7. Copie o **Client ID** e **Client Secret** gerados

Passo 2: Configurar no Supabase Dashboard

1. No Supabase Dashboard, vá em **Authentication → Providers**
2. Localize **Google** na lista e clique em "Enable"
3. Cole o **Client ID** e **Client Secret** do Google
4. Clique em "Save"

4.2. Configurando GitHub OAuth no Supabase

Passo 1: Criar OAuth App no GitHub

1. Acesse GitHub Developer Settings
2. Clique em **New OAuth App**
3. Preencha:
    - Application name: Nome do seu app
    - Homepage URL: `https://seu-dominio.com` (ou deixe `http://localhost` para dev)
    - Authorization callback URL: `https://seu-projeto.supabase.co/auth/v1/callback`
4. Clique em "Register application"
5. Copie o **Client ID**
6. Clique em "Generate a new client secret" e copie o **Client Secret**

Passo 2: Configurar no Supabase Dashboard

1. No Supabase Dashboard, vá em **Authentication → Providers**
2. Localize **GitHub** e clique em "Enable"
3. Cole o **Client ID** e **Client Secret** do GitHub
4. Clique em "Save"

4.3. Implementando Login Social no Flutter

```dart
// lib/services/supabase_auth_service.dart

/// Faz login com provedor OAuth (Google, GitHub, etc)
///
/// [provider]: Provedor OAuth (Provider.google, Provider.github, etc)
///
/// Retorna true se login iniciado com sucesso
/// O fluxo continua no navegador e retorna via deep link
Future<bool> signInWithOAuth(OAuthProvider provider) async {
  try {
    final result = await _supabase.auth.signInWithOAuth(
      provider,
      redirectTo: 'io.supabase.meuapp://login-callback',

      // Configurações adicionais para OAuth
      authScreenLaunchMode: LaunchMode.externalApplication,
    );

    // signInWithOAuth retorna true se o browser foi aberto
    return result;

  } on AuthException catch (e) {
    throw _handleAuthException(e);
  } catch (e) {
    throw Exception('Erro ao iniciar login social: $e');
  }
}
```

O fluxo do OAuth funciona assim:

1. App chama `signInWithOAuth(Provider.google)`
2. Supabase abre navegador externo
3. Usuário faz login no Google/GitHub

4. Google/GitHub redireciona de volta para Supabase
5. Supabase valida e gera tokens
6. Supabase redireciona para seu app via deep link (`io.supabase.meuapp://login-callback`)
7. App captura o deep link e processa o callback
8. Sessão é estabelecida automaticamente

Tela com botões de login social:

```dart
```

```dart
// lib/screens/login_screen.dart (adicionar aos botões)

class _LoginScreenState extends ConsumerState<LoginScreen> {
  // ... código existente ...

  Future<void> _handleGoogleSignIn() async {
    try {
      final authService = SupabaseAuthService();

      final launched = await authService.signInWithOAuth(OAuthProvider.google);

      if (!launched) {
        throw Exception('Não foi possível abrir o navegador');
      }

      // Não precisa fazer nada aqui - o callback será tratado pelo deep link
      // O AuthNotifier (Parte 5) vai detectar a mudança de autenticação

    } catch (e) {
      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Erro no login com Google: $e'),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  Future<void> _handleGitHubSignIn() async {
    try {
      final authService = SupabaseAuthService();

      final launched = await authService.signInWithOAuth(OAuthProvider.github);

      if (!launched) {
        throw Exception('Não foi possível abrir o navegador');
      }

    } catch (e) {
      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Erro no login com GitHub: $e'),
```

```
        backgroundColor: Colors.red,
      ),
    );
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Login')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // ... campos de email/senha existentes ...

            const SizedBox(height: 16),

            // Divisor
            Row(
              children: const [
                Expanded(child: Divider()),
                Padding(
                  padding: EdgeInsets.symmetric(horizontal: 16),
                  child: Text('OU'),
                ),
                Expanded(child: Divider()),
              ],
            ),
            const SizedBox(height: 16),

            // Botão Google
            SizedBox(
              width: double.infinity,
              height: 48,
              child: OutlinedButton.icon(
                onPressed: _handleGoogleSignIn,
                icon: Image.asset(
                  'assets/google_logo.png', // Adicione o logo do Google
                  height: 24,
                ),
                label: const Text('Continuar com Google'),
                style: OutlinedButton.styleFrom(
                  side: BorderSide(color: Colors.grey[300]!),
```

```
            ),
          ),
        ),
        const SizedBox(height: 12),

        // Botão GitHub
        SizedBox(
          width: double.infinity,
          height: 48,
          child: OutlinedButton.icon(
            onPressed: _handleGitHubSignIn,
            icon: const Icon(Icons.code, color: Colors.black),
            label: const Text('Continuar com GitHub'),
            style: OutlinedButton.styleFrom(
              side: BorderSide(color: Colors.grey[300]!),
            ),
          ),
        ),
      ],
    ),
  ),
),
);
}
}
```

4.4. Tratando Callbacks OAuth

O Supabase SDK trata automaticamente os callbacks OAuth! Você não precisa escrever código extra para processar o deep link - o SDK faz isso internamente.

No entanto, é bom estar ciente do que acontece nos bastidores:

1. Quando o usuário completa o OAuth no navegador, o Supabase redireciona para `io.supabase.meuapp://login-callback?access_token=...&refresh_token=...`
2. O sistema operacional abre seu app (graças ao deep linking configurado)
3. O Supabase SDK detecta os parâmetros na URL
4. O SDK automaticamente:
    - Extrai os tokens
    - Armazena de forma segura
    - Estabelece a sessão
    - Dispara o `onAuthStateChange` listener

⚠️ **Troubleshooting OAuth:**

Se o OAuth não funcionar:

1. Verifique se o deep linking está configurado corretamente (AndroidManifest.xml e Info.plist)
2. Confirme que as Redirect URLs no Supabase Dashboard incluem seu esquema: `io.supabase.meuapp://login-callback`
3. Verifique se as credenciais OAuth (Client ID/Secret) estão corretas no dashboard
4. No Google Cloud Console, certifique-se que a URL de callback aponta para `https://seu-projeto.supabase.co/auth/v1/callback`
5. Teste em dispositivo físico (não emulador) se possível - deep linking pode ter comportamento diferente

---

Parte 5: Magic Links (Autenticação Passwordless)

Magic links são emails com links especiais que autenticam o usuário sem necessidade de senha. É uma experiência de usuário excelente e mais segura que senhas fracas.

5.1. Como Funcionam Magic Links

1. Usuário digita apenas o email
2. Supabase envia um email com link único e temporário
3. Usuário clica no link
4. App abre via deep link
5. Sessão é estabelecida automaticamente

5.2. Implementando Magic Links

```dart
```

```dart
// lib/services/supabase_auth_service.dart

/// Envia magic link para email
///
/// [email]: Email do usuário
/// [shouldCreateUser]: Se true, cria usuário se não existir; se false, só funciona para usuários existentes
///
/// Retorna true se email foi enviado
Future<void> signInWithMagicLink({
  required String email,
  bool shouldCreateUser = true,
}) async {
  try {
    await _supabase.auth.signInWithOtp(
      email: email,

      // Configurações do magic link
      emailRedirectTo: 'io.supabase.meuapp://login-callback',

      // Se deve criar usuário se não existir
      // true = funciona como "sign up or sign in"
      // false = só funciona se usuário já existir
      shouldCreateUser: shouldCreateUser,
    );

    // Email enviado com sucesso
    // O Supabase não retorna erro mesmo se email não existir (segurança)

  } on AuthException catch (e) {
    throw _handleAuthException(e);
  } catch (e) {
    throw Exception('Erro ao enviar magic link: $e');
  }
}
```

Tela de Magic Link:

dart

```dart
// lib/screens/magic_link_screen.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../services/supabase_auth_service.dart';

class MagicLinkScreen extends ConsumerStatefulWidget {
  const MagicLinkScreen({super.key});

  @override
  ConsumerState<MagicLinkScreen> createState() => _MagicLinkScreenState();
}

class _MagicLinkScreenState extends ConsumerState<MagicLinkScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();

  bool _isLoading = false;
  bool _emailSent = false;

  @override
  void dispose() {
    _emailController.dispose();
    super.dispose();
  }

  Future<void> _handleSendMagicLink() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => _isLoading = true);

    try {
      final authService = SupabaseAuthService();

      await authService.signInWithMagicLink(
        email: _emailController.text.trim(),
        shouldCreateUser: true, // Permite cadastro automático
      );

      if (!mounted) return;

      setState(() => _emailSent = true);

    } catch (e) {
      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
```

```
      SnackBar(
        content: Text(e.toString()),
        backgroundColor: Colors.red,
      ),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Login sem Senha')),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: _emailSent ? _buildSuccessView() : _buildFormView(),
    ),
  );
}

Widget _buildFormView() {
  return Form(
    key: _formKey,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Icon(
          Icons.email,
          size: 64,
          color: Colors.blue,
        ),
        const SizedBox(height: 24),

        Text(
          'Login Mágico ✨ ',
          style: Theme.of(context).textTheme.headlineSmall,
        ),
        const SizedBox(height: 8),

        const Text(
          'Digite seu email e enviaremos um link de acesso direto. Sem senha necessária!',
          textAlign: TextAlign.center,
          style: TextStyle(color: Colors.grey),
        ),
```

```
        const SizedBox(height: 32),

        TextFormField(
          controller: _emailController,
          decoration: const InputDecoration(
            labelText: 'Email',
            border: OutlineInputBorder(),
            prefixIcon: Icon(Icons.email),
          ),
          keyboardType: TextInputType.emailAddress,
          validator: (value) {
            if (value == null || !value.contains('@')) {
              return 'Digite um email válido';
            }
            return null;
          },
        ),
        const SizedBox(height: 24),

        SizedBox(
          width: double.infinity,
          height: 48,
          child: ElevatedButton(
            onPressed: _isLoading ? null : _handleSendMagicLink,
            child: _isLoading
                ? const CircularProgressIndicator(color: Colors.white)
                : const Text('Enviar Link Mágico'),
          ),
        ),
        const SizedBox(height: 16),

        TextButton(
          onPressed: () {
            Navigator.of(context).pushReplacementNamed('/login');
          },
          child: const Text('Voltar ao login tradicional'),
        ),
      ],
    ),
  );
}

Widget _buildSuccessView() {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      const Icon(
```

```
  Icons.mark_email_read,
  size: 64,
  color: Colors.green,
),
const SizedBox(height: 24),

Text(
  'Email Enviado!',
  style: Theme.of(context).textTheme.headlineSmall,
),
const SizedBox(height: 8),

Text(
  'Enviamos um link de acesso para\n${_emailController.text}',
  textAlign: TextAlign.center,
  style: const TextStyle(color: Colors.grey),
),
const SizedBox(height: 16),

const Text(
  'Clique no link no email para entrar automaticamente.',
  textAlign: TextAlign.center,
  style: TextStyle(fontSize: 12, color: Colors.grey),
),
const SizedBox(height: 8),

const Text(
  'O link expira em 1 hora.',
  textAlign: TextAlign.center,
  style: TextStyle(fontSize: 12, color: Colors.orange),
),
const SizedBox(height: 32),

ElevatedButton(
  onPressed: () {
    setState(() => _emailSent = false);
  },
  child: const Text('Enviar Novamente'),
),
const SizedBox(height: 8),

TextButton(
  onPressed: () {
    Navigator.of(context).pushReplacementNamed('/login');
  },
  child: const Text('Voltar ao Login'),
),
```

```
      ],
    );
  }
}
```

5.3. Configurando Template de Email do Magic Link

Por padrão, o Supabase envia um email genérico. Você pode customizar o template:

1. No Supabase Dashboard, vá em **Authentication → Email Templates**
2. Clique em **Magic Link**
3. Personalize o HTML e o conteúdo
4. Use variáveis disponíveis:
   - `{{ .SiteURL }}`: URL do seu site
   - `{{ .Token }}`: Token de autenticação
   - `{{ .TokenHash }}`: Hash do token
   - `{{ .RedirectTo }}`: URL de redirecionamento

Exemplo de template customizado:

```html
<h2>Seu link de acesso está pronto!</h2>
<p>Olá! Clique no botão abaixo para acessar sua conta:</p>
<p>
  <a href="{{ .SiteURL }}/auth/v1/verify?token={{ .TokenHash }}&type=magiclink&redirect_to={{ .RedirectTo }}"
    style="background: #24b47e; color: white; padding: 12px 24px; text-decoration: none; border-radius: 5px;">
    Acessar Agora
  </a>
</p>
<p>Este link expira em 1 hora e só pode ser usado uma vez.</p>
<p>Se você não solicitou este email, pode ignorá-lo com segurança.</p>
```

Parte 6: Estado de Autenticação e Listeners

O Supabase fornece um stream (`onAuthStateChange`) que notifica toda vez que o estado de autenticação muda. Isso é **essencial** para manter seu app sincronizado.

6.1. Eventos de Autenticação

O Supabase dispara eventos para:

- `SIGNED_IN`: Usuário fez login
- `SIGNED_OUT`: Usuário fez logout
- `TOKEN_REFRESHED`: Token foi renovado automaticamente
- `USER_UPDATED`: Dados do usuário foram atualizados
- `PASSWORD_RECOVERY`: Usuário iniciou recuperação de senha
- `USER_DELETED`: Usuário foi deletado

6.2. Integrando com AuthNotifier do Riverpod

Vamos integrar o `onAuthStateChange` com o AuthNotifier que criamos na Seção 7.

Primeiro, vamos atualizar nosso modelo de estado:

```dart
```

```dart
// lib/models/auth_state.dart
import 'package:supabase_flutter/supabase_flutter.dart';

/// Representa o estado de autenticação do app
sealed class AuthState {
  const AuthState();
}

/// Estado inicial - verificando se há sessão salva
class AuthStateInitial extends AuthState {
  const AuthStateInitial();
}

/// Estado de carregamento (fazendo login, cadastro, etc)
class AuthStateLoading extends AuthState {
  const AuthStateLoading();
}

/// Usuário autenticado
class AuthStateAuthenticated extends AuthState {
  final User user;
  final Session session;

  const AuthStateAuthenticated({
    required this.user,
    required this.session,
  });

  // Helpers para acessar dados comuns
  String get email => user.email ?? '';
  String get id => user.id;
  Map<String, dynamic> get metadata => user.userMetadata ?? {};
}

/// Usuário não autenticado
class AuthStateUnauthenticated extends AuthState {
  const AuthStateUnauthenticated();
}

/// Estado de erro
class AuthStateError extends AuthState {
  final String message;

  const AuthStateError(this.message);
}
```

Agora, o AuthNotifier integrado com Supabase:

dart

```dart
// lib/providers/auth_provider.dart
import 'dart:async';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import '../models/auth_state.dart';
import '../main.dart';
import '../services/supabase_auth_service.dart';

/// Provider do AuthNotifier
final authProvider = StateNotifierProvider<AuthNotifier, AuthState>((ref) {
  return AuthNotifier();
});

/// Gerencia o estado de autenticação integrado com Supabase
class AuthNotifier extends StateNotifier<AuthState> {
  final SupabaseAuthService _authService = SupabaseAuthService();
  StreamSubscription<AuthState>? _authSubscription;

  AuthNotifier() : super(const AuthStateInitial()) {
    _initialize();
  }

  /// Inicializa o notifier e configura listeners
  void _initialize() {
    // Verificar sessão existente
    _checkInitialSession();

    // Escutar mudanças de autenticação
    _setupAuthListener();
  }

  /// Verifica se há uma sessão salva ao iniciar o app
  void _checkInitialSession() {
    final session = supabase.auth.currentSession;

    if (session != null && session.user != null) {
      // Há uma sessão válida salva
      state = AuthStateAuthenticated(
        user: session.user!,
        session: session,
      );
    } else {
      // Nenhuma sessão
      state = const AuthStateUnauthenticated();
    }
  }
```

```dart
/// Configura listener para mudanças de autenticação
void _setupAuthListener() {
  _authSubscription = supabase.auth.onAuthStateChange.listen((data) {
    final event = data.event;
    final session = data.session;

    switch (event) {
      case AuthChangeEvent.signedIn:
        // Usuário fez login
        if (session != null && session.user != null) {
          state = AuthStateAuthenticated(
            user: session.user!,
            session: session,
          );
        }
        break;

      case AuthChangeEvent.signedOut:
        // Usuário fez logout
        state = const AuthStateUnauthenticated();
        break;

      case AuthChangeEvent.tokenRefreshed:
        // Token foi renovado automaticamente
        // Atualizar estado com nova sessão
        if (session != null && session.user != null) {
          state = AuthStateAuthenticated(
            user: session.user!,
            session: session,
          );
        }
        break;

      case AuthChangeEvent.userUpdated:
        // Dados do usuário foram atualizados
        if (session != null && session.user != null) {
          state = AuthStateAuthenticated(
            user: session.user!,
            session: session,
          );
        }
        break;

      case AuthChangeEvent.passwordRecovery:
        // Usuário clicou no link de recuperação de senha
        // Manter como autenticado temporariamente para permitir troca de senha
```

```
        if (session != null && session.user != null) {
          state = AuthStateAuthenticated(
            user: session.user!,
            session: session,
          );
        }
        break;

      default:
        // Outros eventos
        break;
    }
  });
}

/// Login com email e senha
Future<void> signIn({
  required String email,
  required String password,
}) async {
  state = const AuthStateLoading();

  try {
    final response = await _authService.signIn(
      email: email,
      password: password,
    );

    // O listener já vai atualizar o estado automaticamente
    // Mas podemos atualizar manualmente também se preferir:
    // state = AuthStateAuthenticated(user: response.user!, session: response.session!);

  } catch (e) {
    state = AuthStateError(e.toString());
    // Voltar para unauthenticated após mostrar erro
    await Future.delayed(const Duration(seconds: 2));
    state = const AuthStateUnauthenticated();
  }
}

/// Cadastro com email e senha
Future<void> signUp({
  required String email,
  required String password,
  Map<String, dynamic>? metadata,
}) async {
  state = const AuthStateLoading();
```

```
    try {
      final response = await _authService.signUp(
        email: email,
        password: password,
        data: metadata,
      );

      // Se confirmação de email está habilitada, usuário não é autenticado imediatamente
      if (response.user != null && response.user!.emailConfirmedAt == null) {
        state = const AuthStateUnauthenticated();
      }
      // Senão, o listener já vai atualizar o estado

    } catch (e) {
      state = AuthStateError(e.toString());
      await Future.delayed(const Duration(seconds: 2));
      state = const AuthStateUnauthenticated();
    }
  }

  /// Logout
  Future<void> signOut() async {
    try {
      await _authService.signOut();
      // O listener já vai atualizar o estado automaticamente

    } catch (e) {
      state = AuthStateError(e.toString());
    }
  }

  /// Login com OAuth
  Future<void> signInWithOAuth(OAuthProvider provider) async {
    try {
      await _authService.signInWithOAuth(provider);
      // O callback será tratado automaticamente e o listener vai atualizar o estado

    } catch (e) {
      state = AuthStateError(e.toString());
    }
  }

  /// Magic Link
  Future<void> signInWithMagicLink(String email) async {
    state = const AuthStateLoading();
```

```
    try {
      await _authService.signInWithMagicLink(email: email);
      // Voltar para unauthenticated - usuário precisa clicar no link
      state = const AuthStateUnauthenticated();

    } catch (e) {
      state = AuthStateError(e.toString());
      await Future.delayed(const Duration(seconds: 2));
      state = const AuthStateUnauthenticated();
    }
  }

  /// Recuperação de senha
  Future<void> resetPassword(String email) async {
    try {
      await _authService.resetPasswordForEmail(email);

    } catch (e) {
      state = AuthStateError(e.toString());
    }
  }

  /// Atualizar senha (após recuperação)
  Future<void> updatePassword(String newPassword) async {
    try {
      await _authService.updatePassword(newPassword);

      // Após trocar senha, fazer logout para usuário fazer login novamente
      await signOut();

    } catch (e) {
      state = AuthStateError(e.toString());
    }
  }

  @override
  void dispose() {
    _authSubscription?.cancel();
    super.dispose();
  }
}

/// Provider helper para verificar se está autenticado
final isAuthenticatedProvider = Provider<bool>((ref) {
  final authState = ref.watch(authProvider);
  return authState is AuthStateAuthenticated;
});
```

```dart
/// Provider helper para acessar o usuário atual
final currentUserProvider = Provider<User?>((ref) {
  final authState = ref.watch(authProvider);
  return authState is AuthStateAuthenticated ? authState.user : null;
});
```

6.3. Usando o AuthNotifier nas Telas

Auth Wrapper - decide qual tela mostrar baseado no estado:

dart

```dart
// lib/widgets/auth_wrapper.dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../providers/auth_provider.dart';
import '../models/auth_state.dart';
import '../screens/login_screen.dart';
import '../screens/home_screen.dart';

/// Widget que decide qual tela mostrar baseado no estado de autenticação
class AuthWrapper extends ConsumerWidget {
  const AuthWrapper({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final authState = ref.watch(authProvider);

    return switch (authState) {
      // Estado inicial - mostra splash screen
      AuthStateInitial() => const Scaffold(
          body: Center(
            child: CircularProgressIndicator(),
          ),
        ),

      // Carregando - mostra indicador
      AuthStateLoading() => const Scaffold(
          body: Center(
            child: CircularProgressIndicator(),
          ),
        ),

      // Autenticado - mostra app
      AuthStateAuthenticated() => const HomeScreen(),

      // Não autenticado - mostra login
      AuthStateUnauthenticated() => const LoginScreen(),

      // Erro - mostra login com mensagem de erro
      AuthStateError(:final message) => Scaffold(
          body: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                const Icon(Icons.error, size: 48, color: Colors.red),
                const SizedBox(height: 16),
                Text(message),
```

```
            const SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                // Resetar para unauthenticated
                ref.read(authProvider.notifier).signOut();
              },
              child: const Text('Tentar Novamente'),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Atualizar tela de login para usar o provider:

dart

```
// lib/screens/login_screen.dart (atualizado)
Future<void> _handleLogin() async {
 if (!_formKey.currentState!.validate()) return;

 // Usar o AuthNotifier em vez de chamar o service diretamente
 await ref.read(authProvider.notifier).signIn(
  email: _emailController.text.trim(),
  password: _passwordController.text,
 );

 // Não precisa navegar manualmente - o AuthWrapper vai fazer isso
 // baseado na mudança de estado
}
```

Parte 7: Customizações Avançadas

7.1. Templates de Email Personalizados

Você pode customizar todos os templates de email no dashboard:

1. Confirm Signup Email

- Usado quando usuário se cadastra
- Template padrão pede para confirmar email

2. Magic Link Email

- Usado para autenticação sem senha
- Template padrão tem um botão de acesso

3. Reset Password Email

- Usado para recuperação de senha
- Template padrão tem link para redefinir

Como customizar:

1. Vá em **Authentication → Email Templates**
2. Selecione o template
3. Edite o HTML
4. Use variáveis Supabase (ver seção 5.3)
5. Teste enviando email para você mesmo

7.2. Metadados de Usuário

Você pode armazenar informações adicionais no perfil do usuário:

```dart
```

```dart
// Ao cadastrar
await authService.signUp(
  email: 'usuario@email.com',
  password: 'senha123',
  data: {
    'name': 'João Silva',
    'avatar_url': 'https://...',
    'age': 25,
    'preferences': {
      'theme': 'dark',
      'language': 'pt-BR',
    },
  },
);

// Acessar metadados
final user = supabase.auth.currentUser;
final name = user?.userMetadata?['name'];
final preferences = user?.userMetadata?['preferences'];

// Atualizar metadados
await supabase.auth.updateUser(
  UserAttributes(
    data: {
      'name': 'João Pedro Silva',
      'avatar_url': 'https://novo-avatar.com/foto.jpg',
    },
  ),
);
```

7.3. SMTP Customizado

Por padrão, o Supabase usa seu próprio servidor SMTP para enviar emails. Para usar seu próprio:

1. Vá em **Project Settings → Auth**
2. Role até **SMTP Settings**
3. Habilite "Enable Custom SMTP"
4. Configure:
   - **Host**: Ex: `smtp.gmail.com`
   - **Port**: Ex: `587` (TLS) ou `465` (SSL)
   - **Username**: Seu email
   - **Password**: Senha de app (não a senha normal!)

- **Sender Name**: Nome que aparece no "De:"
- **Sender Email**: Email remetente

Para Gmail:

1. Ative verificação em 2 etapas
2. Crie uma "Senha de app" em https://myaccount.google.com/apppasswords
3. Use essa senha nas configurações SMTP

7.4. Row Level Security (RLS) Básico

RLS é como você protege dados no banco de dados Supabase. É um tópico avançado, mas aqui está o básico:

Exemplo: Tabela de perfis de usuário

```sql
```

```sql
-- Criar tabela
CREATE TABLE profiles (
  id UUID REFERENCES auth.users PRIMARY KEY,
  email TEXT,
  name TEXT,
  avatar_url TEXT,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Habilitar RLS
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;

-- Política: Usuários podem ver apenas seu próprio perfil
CREATE POLICY "Users can view own profile"
  ON profiles
  FOR SELECT
  USING (auth.uid() = id);

-- Política: Usuários podem atualizar apenas seu próprio perfil
CREATE POLICY "Users can update own profile"
  ON profiles
  FOR UPDATE
  USING (auth.uid() = id);

-- Política: Usuários podem inserir apenas seu próprio perfil
CREATE POLICY "Users can insert own profile"
  ON profiles
  FOR INSERT
  WITH CHECK (auth.uid() = id);
```

Agora, mesmo que um usuário tente acessar o perfil de outro via API, o RLS vai bloquear.

Parte 8: Troubleshooting Comum

8.1. Deep Linking Não Funciona

Problema: Após clicar no magic link ou OAuth, o app não abre.

Soluções:

1. Verificar AndroidManifest.xml e Info.plist
2. Confirmar que o esquema (`io.supabase.meuapp`) é único
3. Testar com comando ADB:

```bash
adb shell am start -a android.intent.action.VIEW -d "io.supabase.meuapp://login-callback"
```

4. No iOS, confirmar que Info.plist tem CFBundleURLSchemes correto
5. Rebuild completo do app após mudar configurações

8.2. "Invalid login credentials"

Problema: Login falha mesmo com credenciais corretas.

Causas possíveis:

1. Email não confirmado (se confirmação estiver habilitada)
   - Solução: Desabilitar confirmação temporariamente ou pedir usuário para confirmar
2. Senha incorreta
3. Usuário foi deletado no dashboard
4. Projeto Supabase em modo pausado (tier gratuito após inatividade)

8.3. Tokens Expirados

Problema: Usuário perde sessão frequentemente.

Soluções:

1. Verificar configurações de expiração no dashboard: **Authentication → Settings → JWT Expiry**
2. O Supabase renova tokens automaticamente - verificar se o listener está configurado
3. Verificar se flutter_secure_storage está funcionando (tokens são salvos lá)

8.4. OAuth não redireciona de volta

Problema: Após login no Google/GitHub, não volta para o app.

Soluções:

1. Verificar se Redirect URI no Google Cloud / GitHub está correto:
   - Deve ser: `https://seu-projeto.supabase.co/auth/v1/callback`
2. Verificar se Redirect URLs no Supabase incluem o esquema do app
3. Testar OAuth em navegador normal primeiro (sem app) para isolar problema

8.5. "Client ID or Client Secret is invalid"

Problema: OAuth falha com erro de credenciais.

Soluções:

1. Recopiar Client ID e Secret do Google Cloud / GitHub
2. Verificar se não há espaços extras nas credenciais
3. Garantir que OAuth App está ativo no GitHub
4. Confirmar que consent screen está configurado no Google

Conclusão da Seção

Parabéns! Você agora sabe como implementar um sistema de autenticação completo com Supabase no Flutter, incluindo:

✅ Configuração do projeto Supabase
✅ Integração do SDK no Flutter
✅ Cadastro, login e logout tradicionais
✅ Recuperação e redefinição de senha
✅ Login social com OAuth (Google, GitHub)
✅ Magic links (autenticação sem senha)
✅ Gerenciamento de estado com Riverpod
✅ Listeners de mudanças de autenticação
✅ Deep linking para callbacks
✅ Customizações avançadas
✅ Troubleshooting de problemas comuns

Próximos passos:

Na próxima seção, vamos criar protótipos visuais detalhados usando Material Design 3 e fornecer prompts para gerar interfaces com IA. Depois disso, você terá desafios práticos para consolidar todo o aprendizado!

Recursos adicionais:

- Documentação oficial Supabase Flutter
- Guia de autenticação Supabase
- Exemplos de código

Fim da Seção 11: Autenticação com Supabase - Guia Completo