Seção 14: Erros Comuns e Como Evitá-los

Desenvolvimento de Aplicativos Móveis - Flutter

Introdução

Esta é a última seção do material didático sobre autenticação em Flutter. Aqui você encontrará os **erros mais comuns** cometidos por desenvolvedores ao implementar autenticação, junto com **soluções práticas** e **exemplos de código** mostrando a forma errada e a forma correta.

Esta seção é essencial porque:

- **Economiza tempo** - Você evitará horas de debugging
- **Previne bugs em produção** - Muitos desses erros só aparecem em situações reais
- **Melhora a segurança** - Alguns erros podem comprometer a segurança dos usuários
- **Ensina boas práticas** - Mostra não apenas o que evitar, mas como fazer corretamente

Como usar esta seção:

1. **Antes de começar seu projeto**: Leia todos os erros para estar ciente
2. **Durante o desenvolvimento**: Consulte quando encontrar problemas
3. **Code review**: Use como checklist para revisar código
4. **Debugging**: Procure o erro específico que você está enfrentando

Vamos aos 15 erros mais comuns!

Erro #1: Não Validar Dados Antes de Enviar

🔴 Sintoma

- App trava quando usuário tenta fazer login com campos vazios
- Supabase retorna erro que não é tratado
- UX ruim - erro aparece depois de delay de rede

🔍 Causa

Enviar dados para o backend sem validar no cliente primeiro.

❌ Código Errado

dart

```dart
// Sem validação - envia direto para o Supabase
Future<void> _handleLogin() async {
  try {
    final email = _emailController.text; // Pode estar vazio!
    final password = _passwordController.text; // Pode estar vazio!

    // Envia sem validar
    await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );

    Navigator.pushReplacement(context, ...);
  } catch (e) {
    // Erro genérico
    print('Erro: $e');
  }
}
```

✅ Código Correto

dart

```dart
// Com validação apropriada
final _formKey = GlobalKey<FormState>();

Future<void> _handleLogin() async {
  // 1. Validar ANTES de enviar
  if (!_formKey.currentState!.validate()) {
    return; // Para se validação falhar
  }

  setState(() => _isLoading = true);

  try {
    final email = _emailController.text.trim(); // Remove espaços
    final password = _passwordController.text;

    await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );

    if (!mounted) return;
    Navigator.pushReplacement(context, ...);

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

    // Mensagem específica baseada no erro
    final message = e.message.contains('Invalid')
        ? 'Email ou senha incorretos'
        : 'Erro ao fazer login: ${e.message}';

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  } finally {
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}

// Validators no Form
TextFormField(
  controller: _emailController,
  validator: (value) {
    if (value == null || value.trim().isEmpty) {
```

```
      return 'Digite seu email';
    }
    if (!value.contains('@')) {
      return 'Email inválido';
    }
    return null;
  },
)
```

📝 Explicação

Por que isso importa:

- Validação client-side é instantânea (sem delay de rede)
- Economiza chamadas desnecessárias ao servidor
- Melhora a UX com feedback imediato
- Reduz custos (menos requisições)

Regra de ouro: Sempre valide no cliente E no servidor. Validação client-side é para UX, validação server-side é para segurança.

Erro #2: Não Tratar Erros de Rede

🔴 Sintoma

- App trava quando internet cai
- Spinner de loading fica girando infinitamente
- Usuário não sabe o que aconteceu

🔍 Causa

Não capturar exceções de rede (timeout, sem conexão, etc).

❌ Código Errado

dart

```dart
Future<void> signIn(String email, String password) async {
  // Não trata timeout ou falta de conexão
  final response = await supabase.auth.signInWithPassword(
    email: email,
    password: password,
  );

  return response; // E se a rede cair?
}
```

✅ Código Correto

dart

```dart
Future<AuthResponse> signIn(String email, String password) async {
  try {
    final response = await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    ).timeout(
      const Duration(seconds: 15), // Timeout explícito
      onTimeout: () {
        throw TimeoutException('A operação demorou muito tempo');
      },
    );

    return response;

  } on AuthException catch (e) {
    // Erro de autenticação do Supabase
    throw _handleAuthException(e);

  } on TimeoutException catch (_) {
    // Timeout específico
    throw Exception(
      'Tempo esgotado. Verifique sua conexão e tente novamente.',
    );

  } on SocketException catch (_) {
    // Sem internet
    throw Exception(
      'Sem conexão com a internet. Conecte-se e tente novamente.',
    );

  } catch (e) {
    // Outros erros não previstos
    throw Exception('Erro inesperado: ${e.toString()}');
  }
}

// Na UI, tratar diferentes tipos de erro
Future<void> _handleLogin() async {
  try {
    await authService.signIn(email, password);
    // Sucesso
  } catch (e) {
    // Mostrar erro apropriado ao usuário
    final message = e.toString().replaceFirst('Exception: ', '');

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

```
      SnackBar(
        content: Text(message),
        action: e.toString().contains('conexão')
            ? SnackBarAction(
                label: 'Tentar Novamente',
                onPressed: _handleLogin,
              )
            : null,
      ),
    );
  }
}
```

📝 Explicação

Tipos de erro a tratar:

1. **AuthException** - Credenciais inválidas, email não confirmado
2. **TimeoutException** - Operação demorou muito
3. **SocketException** - Sem internet
4. **Outros** - Erros não previstos

Sempre:

- Use timeout explícito (padrão pode ser muito longo)
- Dê feedback específico ao usuário
- Ofereça ação de retry quando apropriado

Erro #3: Armazenar Dados Sensíveis de Forma Insegura

🔴 Sintoma

- Credenciais ou tokens em SharedPreferences
- Possível acesso não autorizado aos dados
- Violação de boas práticas de segurança

🔍 Causa

Usar SharedPreferences ou similar para dados sensíveis.

❌ Código Errado

dart

```dart
// NUNCA FAÇA ISSO!
class AuthStorage {
  final SharedPreferences _prefs;

  // Armazenando senha em texto plano - PÉSSIMO!
  Future<void> saveCredentials(String email, String password) async {
    await _prefs.setString('email', email);
    await _prefs.setString('password', password); // 🧯 NUNCA!
  }

  // Armazenando token sem criptografia - RUIM!
  Future<void> saveToken(String token) async {
    await _prefs.setString('auth_token', token); // SharedPreferences não é seguro
  }
}
```

✅ Código Correto

dart

```dart
// Use flutter_secure_storage ou deixe o Supabase gerenciar
class SecureAuthStorage {
  final FlutterSecureStorage _storage = const FlutterSecureStorage();

  // NUNCA armazene senhas - mesmo com criptografia!
  // Use tokens que podem ser revogados

  // Armazenar token de forma segura (mas Supabase já faz isso!)
  Future<void> saveToken(String token) async {
    await _storage.write(
      key: 'auth_token',
      value: token,
      // flutter_secure_storage usa Keychain (iOS) e KeyStore (Android)
    );
  }

  Future<String?> getToken() async {
    return await _storage.read(key: 'auth_token');
  }

  Future<void> deleteToken() async {
    await _storage.delete(key: 'auth_token');
  }
}

// MELHOR AINDA: Deixe o Supabase gerenciar
// O Supabase SDK automaticamente armazena tokens de forma segura
// Você não precisa gerenciar manualmente!

void main() async {
  await Supabase.initialize(
    url: supabaseUrl,
    anonKey: supabaseAnonKey,
    // Supabase usa flutter_secure_storage internamente
    // Tokens são salvos e recuperados automaticamente
  );
}

// Para acessar o usuário/sessão:
final session = supabase.auth.currentSession;
final user = supabase.auth.currentUser;
// Não precisa gerenciar tokens manualmente!
```

📝 Explicação

NUNCA armazene:

- ❌ Senhas (mesmo criptografadas!)
- ❌ Dados sensíveis em SharedPreferences
- ❌ API keys no código
- ❌ Tokens em texto plano

Use:

- ✅ flutter_secure_storage para dados sensíveis
- ✅ Deixe o Supabase SDK gerenciar tokens (ele já faz isso!)
- ✅ Tokens JWT (podem ser revogados)
- ✅ Refresh tokens para renovação automática

Por que o Supabase é seguro:

- Usa flutter_secure_storage internamente
- Gerencia refresh automático de tokens
- Tokens expiram e são renovados automaticamente
- Você não precisa se preocupar com detalhes de implementação

---

Erro #4: Não Dar Feedback Visual Durante Operações Assíncronas

🔴 Sintoma

- Usuário clica em "Login" e nada acontece
- Não fica claro se o app travou ou está processando
- Usuários clicam múltiplas vezes no botão

🔍 Causa

Não mostrar loading indicators ou desabilitar botões durante operações.

❌ Código Errado

dart

```
// Sem feedback visual
ElevatedButton(
  onPressed: () async {
    // Operação demorada, mas nenhum feedback
    await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );
  },
  child: const Text('Entrar'),
)
```

✅ Código Correto

dart

```dart
// Com feedback apropriado
class LoginScreen extends StatefulWidget {
  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  bool _isLoading = false;

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

    // Indicar que está carregando
    setState(() => _isLoading = true);

    try {
      await supabase.auth.signInWithPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text,
      );

      if (!mounted) return;
      Navigator.pushReplacement(context, ...);

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

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(e.toString())),
      );
    } finally {
      // Sempre remove loading, mesmo em erro
      if (mounted) {
        setState(() => _isLoading = false);
      }
    }
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        // Campos do formulário...

        const SizedBox(height: 24),
```

```
        // Botão com loading
        SizedBox(
          width: double.infinity,
          height: 48,
          child: ElevatedButton(
            // Desabilita quando está carregando
            onPressed: _isLoading ? null : _handleLogin,
            child: _isLoading
                ? const SizedBox(
                    height: 20,
                    width: 20,
                    child: CircularProgressIndicator(
                      strokeWidth: 2,
                      color: Colors.white,
                    ),
                  )
                : const Text('Entrar'),
          ),
        ),
      ],
    );
  }
}

// Alternativa: Usar overlay de loading na tela inteira
Future<void> _handleLoginWithOverlay() async {
  // Mostrar overlay
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (context) => const Center(
      child: CircularProgressIndicator(),
    ),
  );

  try {
    await supabase.auth.signInWithPassword(...);

    Navigator.of(context).pop(); // Remove overlay
    Navigator.pushReplacement(context, ...);

  } catch (e) {
    Navigator.of(context).pop(); // Remove overlay

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

```
  }
}
```

📝 Explicação

Tipos de feedback:

1. **Button loading** - Spinner dentro do botão
2. **Overlay** - Loading sobre a tela toda
3. **Skeleton screens** - Placeholder animado
4. **Progress bar** - Barra de progresso (se aplicável)

Sempre:

- Desabilite botões durante operações
- Mostre indicador visual
- Use `finally` para garantir que loading seja removido
- Verifique `mounted` antes de chamar `setState`

Erro #5: Não Limpar Controllers e Dispose Resources

🔴 Sintoma

- Memory leaks
- App fica lento com o tempo
- Warnings no console sobre listeners não removidos

🔍 Causa

Esquecer de fazer dispose de controllers, listeners e streams.

❌ Código Errado

dart

```dart
class LoginScreen extends StatefulWidget {
  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  // Controllers criados mas nunca descartados
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  StreamSubscription? _authSubscription;

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

    // Listener criado mas nunca cancelado
    _authSubscription = supabase.auth.onAuthStateChange.listen((data) {
      // Handle auth changes
    });
  }

  // ❌ FALTANDO dispose()!

  @override
  Widget build(BuildContext context) {
    return TextField(controller: _emailController);
  }
}
```

✅ Código Correto

dart

```dart
class LoginScreen extends StatefulWidget {
  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _formKey = GlobalKey<FormState>();
  StreamSubscription<AuthState>? _authSubscription;

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

    _authSubscription = supabase.auth.onAuthStateChange.listen((data) {
      // Handle auth changes
    });
  }

  // ✅ SEMPRE implemente dispose()
  @override
  void dispose() {
    // Descartar controllers
    _emailController.dispose();
    _passwordController.dispose();

    // Cancelar subscriptions
    _authSubscription?.cancel();

    // Sempre chame super.dispose() por último
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        children: [
          TextField(controller: _emailController),
          TextField(controller: _passwordController),
        ],
      ),
    );
  }
```

```dart
}

// Se usar AnimationController
class AnimatedLoginScreen extends StatefulWidget {
  @override
  State<AnimatedLoginScreen> createState() => _AnimatedLoginScreenState();
}

class _AnimatedLoginScreenState extends State<AnimatedLoginScreen>
    with SingleTickerProviderStateMixin {
  late AnimationController _animationController;

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

    _animationController = AnimationController(
      duration: const Duration(milliseconds: 300),
      vsync: this,
    );
  }

  @override
  void dispose() {
    // Descartar AnimationController
    _animationController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

📝 Explicação

Sempre faça dispose de:

- ✅ TextEditingController
- ✅ AnimationController
- ✅ FocusNode
- ✅ StreamSubscription
- ✅ Timer

- ✅ Qualquer recurso que aloque memória

Pattern:

```dart
@override
void dispose() {
  // 1. Cancelar listeners e subscriptions
  _subscription?.cancel();

  // 2. Descartar controllers
  _controller.dispose();

  // 3. Chamar super.dispose() por último
  super.dispose();
}
```

Dica: Use o lint `always_dispose` para detectar isso automaticamente.

Erro #6: Deep Linking Não Funciona

🔴 Sintoma

- Links de confirmação de email não abrem o app
- OAuth redirect não funciona
- Magic links não funcionam

🔍 Causa

Configuração incorreta ou incompleta de deep linking.

❌ Problemas Comuns

1. Esquema incorreto ou não único:

```xml
<!-- AndroidManifest.xml - ERRADO -->
<data
    android:scheme="app"  <!-- Muito genérico! -->
    android:host="login" />
```

2. Faltando configuração no iOS:

xml

```xml
<!-- Info.plist - FALTANDO -->
<!-- Esqueceu de adicionar CFBundleURLTypes completamente -->
```

3. URL no Supabase não corresponde:

dart

```dart
// No código
redirectTo: 'io.supabase.meuapp://callback'

// No Supabase Dashboard Redirect URLs
// FALTANDO: io.supabase.meuapp://callback
```

✅ Código Correto

1. AndroidManifest.xml (completo):

xml

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application>
    <activity
      android:name=".MainActivity"
      android:exported="true"
      android:launchMode="singleTop"
      android:theme="@style/LaunchTheme">

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

      <!-- ✅ Deep linking correto -->
      <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" />

        <!-- Use seu domínio invertido para unicidade -->
        <data
          android:scheme="com.seudominio.seuapp"
          android:host="login-callback" />

        <!-- Adicione outros paths se necessário -->
        <data
          android:scheme="com.seudominio.seuapp"
          android:host="reset-password" />
      </intent-filter>

    </activity>
  </application>
</manifest>
```

2. Info.plist (completo):

```xml
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <!-- Outras configurações... -->

  <!-- ✅ Deep linking configuração -->
  <key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
      <key>CFBundleURLName</key>
      <string>com.seudominio.seuapp</string>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>com.seudominio.seuapp</string>
      </array>
    </dict>
  </array>

  <!-- Para Flutter deep linking -->
  <key>FlutterDeepLinkingEnabled</key>
  <true/>

</dict>
</plist>
```

3. Supabase Dashboard:

Authentication → URL Configuration → Redirect URLs

Adicione:
✅ com.seudominio.seuapp://login-callback
✅ com.seudominio.seuapp://reset-password
✅ com.seudominio.seuapp://magic-link

4. Código Flutter:

dart

```
// main.dart
await Supabase.initialize(
  url: supabaseUrl,
  anonKey: supabaseAnonKey,
  authOptions: const FlutterAuthClientOptions(
    authFlowType: AuthFlowType.pkce,
  ),
);

// Ao enviar email de recuperação
await supabase.auth.resetPasswordForEmail(
  email,
  redirectTo: 'com.seudominio.seuapp://reset-password',
);

// Ao fazer signInWithOAuth
await supabase.auth.signInWithOAuth(
  OAuthProvider.google,
  redirectTo: 'com.seudominio.seuapp://login-callback',
);
```

5. Testar deep linking:

bash

```
# Android (ADB)
adb shell am start -a android.intent.action.VIEW \
  -d "com.seudominio.seuapp://login-callback?access_token=test"

# iOS (Simulator)
xcrun simctl openurl booted \
  "com.seudominio.seuapp://login-callback?access_token=test"
```

📝 Explicação

Checklist de deep linking:

- [ ] Esquema único (use domínio invertido)
- [ ] AndroidManifest.xml configurado
- [ ] Info.plist configurado
- [ ] URLs adicionadas no Supabase Dashboard
- [ ] `redirectTo` corresponde ao esquema configurado
- [ ] Testado com ADB/xcrun
- [ ] Testado em device real

Dica: Sempre teste em device físico, não apenas emulador.

Erro #7: Expor Informações Sensíveis em Logs

🔴 Sintoma

- Senhas aparecem em logs
- Tokens aparecem em stack traces
- Informações pessoais expostas

🔍 Causa

Usar `print()` ou `debugPrint()` com dados sensíveis.

❌ Código Errado

dart

```dart
Future<void> signIn(String email, String password) async {
  // ❌ NUNCA faça isso!
  print('Login attempt: email=$email, password=$password');

  try {
    final response = await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );

    // ❌ Expondo token
    print('Token: ${response.session?.accessToken}');

  } catch (e) {
    // ❌ Stack trace pode conter dados sensíveis
    print('Error: $e');
  }
}
```

✅ Código Correto

dart

```dart
import 'package:logger/logger.dart';

class AuthService {
  final Logger _logger = Logger(
    printer: PrettyPrinter(
      methodCount: 0,
      errorMethodCount: 5,
      lineLength: 80,
      colors: true,
      printEmojis: true,
    ),
    level: kDebugMode ? Level.debug : Level.error,
  );

  Future<void> signIn(String email, String password) async {
    // ✅ Log sem dados sensíveis
    _logger.d('Attempting login for user');

    try {
      final response = await supabase.auth.signInWithPassword(
        email: email,
        password: password, // Não fazer log da senha!
      );

      // ✅ Log apenas informações não sensíveis
      _logger.i('Login successful for user: ${response.user?.id}');

      // Se realmente precisa do token (desenvolvimento), censure
      if (kDebugMode) {
        final token = response.session?.accessToken;
        final censored = token != null
            ? '${token.substring(0, 10)}...${token.substring(token.length - 10)}'
            : 'null';
        _logger.d('Token: $censored');
      }

    } on AuthException catch (e) {
      // ✅ Log erro sem expor dados do usuário
      _logger.e('Login failed', error: e.statusCode);
      rethrow;

    } catch (e, stackTrace) {
      _logger.e('Unexpected error during login', error: e, stackTrace: stackTrace);
      rethrow;
    }
  }
```

```
}

// Para produção, use um serviço de logging apropriado
// como Sentry ou Firebase Crashlytics
void _setupErrorTracking() {
  if (kReleaseMode) {
    FlutterError.onError = (details) {
      // Enviar para Sentry, etc
      // MAS sanitize dados sensíveis primeiro!
    };
  }
}
```

📝 Explicação

NUNCA faça log de:

- ❌ Senhas
- ❌ Tokens (access, refresh)
- ❌ Dados pessoais completos (CPF, RG, etc)
- ❌ Números de cartão
- ❌ Códigos de verificação

Pode fazer log de:

- ✅ User IDs (UUID)
- ✅ Códigos de erro (status codes)
- ✅ Fluxo de execução (entering/leaving functions)
- ✅ Timestamps
- ✅ Tipos de operação (login, signup, etc)

Em produção:

- Use serviços profissionais (Sentry, Crashlytics)
- Implemente sanitização de dados
- Configure níveis de log apropriados
- Revise logs periodicamente

Erro #8: Não Lidar com Sessões Expiradas

🔴 Sintoma

- Usuário perde acesso de repente
- Requisições falham com 401
- App não redireciona para login

🔍 Causa

Não monitorar expiração de tokens ou não renovar automaticamente.

❌ Código Errado

dart

```dart
// Assume que sessão nunca expira
Future<void> fetchUserData() async {
  // ❌ E se o token expirou?
  final response = await supabase
      .from('profiles')
      .select()
      .eq('id', supabase.auth.currentUser!.id);

  // Vai falhar com 401 se sessão expirou
}
```

✅ Código Correto

dart

```dart
// 1. O Supabase SDK já renova tokens automaticamente!
// Basta escutar mudanças de autenticação

class AuthNotifier extends StateNotifier<AuthState> {
  StreamSubscription<AuthState>? _authSubscription;

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

  void _setupAuthListener() {
    _authSubscription = supabase.auth.onAuthStateChange.listen((data) {
      final event = data.event;
      final session = data.session;

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

        case AuthChangeEvent.tokenRefreshed:
          // ✅ Token foi renovado automaticamente!
          if (session != null) {
            state = AuthStateAuthenticated(session.user, session);
          }
          break;

        case AuthChangeEvent.signedOut:
          state = const AuthStateUnauthenticated();
          break;

        default:
          break;
      }
    });
  }

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

```dart
// 2. Para requisições manuais, sempre verifique sessão
Future<List<dynamic>> fetchUserData() async {
  // Verificar se há sessão válida
  final session = supabase.auth.currentSession;

  if (session == null) {
    throw Exception('Usuário não autenticado');
  }

  // Verificar se token está perto de expirar (menos de 5 min)
  final expiresAt = DateTime.fromMillisecondsSinceEpoch(
    session.expiresAt! * 1000,
  );

  if (expiresAt.difference(DateTime.now()).inMinutes < 5) {
    // Forçar renovação
    await supabase.auth.refreshSession();
  }

  // Agora fazer a requisição
  final response = await supabase
      .from('profiles')
      .select()
      .eq('id', session.user.id);

  return response;
}

// 3. Interceptor para renovação automática (se usar Dio)
class AuthInterceptor extends Interceptor {
  @override
  Future<void> onError(
    DioException err,
    ErrorInterceptorHandler handler,
  ) async {
    if (err.response?.statusCode == 401) {
      // Token expirado, tentar renovar
      try {
        await supabase.auth.refreshSession();

        // Retry requisição original
        final options = Options(
          method: err.requestOptions.method,
          headers: err.requestOptions.headers,
        );

        final response = await Dio().request(
```

```
          err.requestOptions.path,
          options: options,
          data: err.requestOptions.data,
          queryParameters: err.requestOptions.queryParameters,
        );

        return handler.resolve(response);

      } catch (e) {
        // Renovação falhou, fazer logout
        await supabase.auth.signOut();
        return handler.reject(err);
      }
    }

    return handler.next(err);
  }
}
```

📝 Explicação

O Supabase já faz muito trabalho:

- ✅ Renova tokens automaticamente (se possível)
- ✅ Dispara evento `tokenRefreshed`
- ✅ Armazena nova sessão automaticamente

Você precisa:

- ✅ Escutar `onAuthStateChange`
- ✅ Atualizar UI quando token é renovado
- ✅ Lidar com casos onde renovação falha (forçar re-login)
- ✅ Verificar validade antes de requisições críticas

Configurações de expiração: No Supabase Dashboard: Authentication → Settings

- JWT Expiry: Quanto tempo o access token dura (padrão 1h)
- Refresh Token Expiry: Quanto tempo o refresh token dura (padrão 30 dias)

Erro #9: URLs Incorretas ou Configurações Erradas

🔴 Sintoma

- Supabase retorna 404
- OAuth não funciona
- Erros de CORS

🔍 Causa

URL do projeto, anon key ou redirect URLs incorretos.

❌ Código Errado

dart

```dart
// ❌ URLs hardcoded e possivelmente erradas
const supabaseUrl = 'https://my-project.supabase.co'; // Pode estar errado
const supabaseAnonKey = 'eyJh...'; // Pode estar expirada ou errada

await Supabase.initialize(
  url: supabaseUrl,
  anonKey: supabaseAnonKey,
);

// ❌ Redirect URL não corresponde à configuração
await supabase.auth.signInWithOAuth(
  OAuthProvider.google,
  redirectTo: 'myapp://callback', // Não configurado no dashboard
);
```

✅ Código Correto

dart

```
// 1. Use variáveis de ambiente (nunca commit credenciais)
// .env file (adicione ao .gitignore!)
/*
SUPABASE_URL=https://xyzabcdefgh.supabase.co
SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
REDIRECT_URL_SCHEME=com.seudominio.seuapp
*/

// Instale flutter_dotenv
dependencies:
  flutter_dotenv: ^5.1.0

// pubspec.yaml
assets:
  - .env

// main.dart
import 'package:flutter_dotenv/flutter_dotenv.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Carregar variáveis de ambiente
  await dotenv.load(fileName: ".env");

  // ✅ Usar variáveis de ambiente
  await Supabase.initialize(
    url: dotenv.env['SUPABASE_URL']!,
    anonKey: dotenv.env['SUPABASE_ANON_KEY']!,
  );

  runApp(const MyApp());
}

// 2. Criar arquivo de configuração
// lib/config/supabase_config.dart
class SupabaseConfig {
  // Valores padrão para desenvolvimento
  static const String defaultUrl = String.fromEnvironment(
    'SUPABASE_URL',
    defaultValue: '', // Vazio força uso de .env
  );

  static const String defaultAnonKey = String.fromEnvironment(
    'SUPABASE_ANON_KEY',
    defaultValue: '',
```

```
  );

  // Validação
  static void validate() {
    if (defaultUrl.isEmpty) {
      throw Exception('SUPABASE_URL não configurada. Verifique seu arquivo .env');
    }

    if (defaultAnonKey.isEmpty) {
      throw Exception('SUPABASE_ANON_KEY não configurada. Verifique seu arquivo .env');
    }

    // Validar formato da URL
    if (!defaultUrl.startsWith('https://') || !defaultUrl.contains('.supabase.co')) {
      throw Exception('SUPABASE_URL inválida. Formato: https://projeto.supabase.co');
    }
  }

  // Redirect URLs
  static String get redirectUrlScheme => dotenv.env['REDIRECT_URL_SCHEME']!;

  static String loginCallback() => '$redirectUrlScheme://login-callback';
  static String resetPassword() => '$redirectUrlScheme://reset-password';
  static String magicLink() => '$redirectUrlScheme://magic-link';
}

// Usar na aplicação
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await dotenv.load(fileName: ".env");

  // Validar configuração
  SupabaseConfig.validate();

  await Supabase.initialize(
    url: SupabaseConfig.defaultUrl,
    anonKey: SupabaseConfig.defaultAnonKey,
  );

  runApp(const MyApp());
}

// Usar em OAuth
await supabase.auth.signInWithOAuth(
  OAuthProvider.google,
  redirectTo: SupabaseConfig.loginCallback(),
```

```
);

// 3. Template .env.example (commitar isso)
/*
# Supabase Configuration
# Get these values from https://app.supabase.com/project/_/settings/api

SUPABASE_URL=https://seu-projeto.supabase.co
SUPABASE_ANON_KEY=sua-anon-key-aqui

# Deep Linking
REDIRECT_URL_SCHEME=com.seudominio.seuapp
*/

// 4. README com instruções
/*
## Setup

1. Copie `.env.example` para `.env`
2. Preencha com suas credenciais do Supabase
3. Execute `flutter pub get`
4. Execute `flutter run`

### Obtendo Credenciais Supabase

1. Acesse https://app.supabase.com
2. Selecione seu projeto
3. Vá em Settings → API
4. Copie Project URL e anon/public key
*/
```

📝 Explicação

Boas práticas para configuração:

1. ✅ Use variáveis de ambiente (.env)
2. ✅ Nunca commit credenciais no git
3. ✅ Adicione .env ao .gitignore
4. ✅ Commit .env.example como template
5. ✅ Valide configurações no startup
6. ✅ Documente no README

Checklist de URLs:

- [ ] URL do Supabase correta (formato: https://projeto.supabase.co)
- [ ] Anon key correta (copie do dashboard)
- [ ] Redirect URLs adicionadas no dashboard
- [ ] Redirect URLs correspondem ao código
- [ ] Esquema de deep link único e consistente

Erro #10: Não Testar Cenários de Erro

🔴 Sintoma

- App trava em situações inesperadas
- Bugs descobertos em produção
- Má experiência do usuário

🔍 Causa

Testar apenas "happy path" (quando tudo funciona).

❌ Abordagem Errada

dart

```dart
// Testa apenas caso de sucesso
test('signIn with valid credentials', () async {
  final result = await authService.signIn(
    'valid@email.com',
    'validpassword',
  );

  expect(result.user, isNotNull);
});

// Esquece de testar:
// - Email inválido
// - Senha errada
// - Sem internet
// - Timeout
// - Campos vazios
// - Email não confirmado
```

✅ Abordagem Correta

dart

```dart
// Teste abrangente com múltiplos cenários
group('Authentication', () {
  late MockSupabaseClient mockSupabase;
  late AuthService authService;

  setUp(() {
    mockSupabase = MockSupabaseClient();
    authService = AuthService(mockSupabase);
  });

  group('signIn', () {
    test('successful login returns user', () async {
      // Arrange
      when(mockSupabase.auth.signInWithPassword(
        email: anyNamed('email'),
        password: anyNamed('password'),
      )).thenAnswer((_) async => AuthResponse(
        user: User(id: '123', email: 'test@example.com'),
        session: Session(...),
      ));

      // Act
      final result = await authService.signIn(
        'test@example.com',
        'password123',
      );

      // Assert
      expect(result.user, isNotNull);
      expect(result.user!.email, 'test@example.com');
    });

    test('invalid credentials throws exception', () async {
      // Arrange
      when(mockSupabase.auth.signInWithPassword(
        email: anyNamed('email'),
        password: anyNamed('password'),
      )).thenThrow(AuthException('Invalid login credentials'));

      // Act & Assert
      expect(
        () => authService.signIn('wrong@email.com', 'wrongpass'),
        throwsA(isA<Exception>()),
      );
    });
```

```
test('network error throws appropriate exception', () async {
  // Arrange
  when(mockSupabase.auth.signInWithPassword(
    email: anyNamed('email'),
    password: anyNamed('password'),
  )).thenThrow(SocketException('No internet'));

  // Act & Assert
  expect(
    () => authService.signIn('test@email.com', 'pass'),
    throwsA(predicate((e) =>
      e.toString().contains('conexão')
    )),
  );
});

test('timeout throws timeout exception', () async {
  // Arrange
  when(mockSupabase.auth.signInWithPassword(
    email: anyNamed('email'),
    password: anyNamed('password'),
  )).thenAnswer((_) async {
    await Future.delayed(const Duration(seconds: 20));
    return AuthResponse(...);
  });

  // Act & Assert
  expect(
    () => authService.signIn('test@email.com', 'pass'),
    throwsA(isA<TimeoutException>()),
  );
});

test('empty email throws validation error', () async {
  // Act & Assert
  expect(
    () => authService.signIn('', 'password'),
    throwsA(predicate((e) =>
      e.toString().contains('email')
    )),
  );
});

test('empty password throws validation error', () async {
  // Act & Assert
  expect(
    () => authService.signIn('test@email.com', ''),
```

```
        throwsA(predicate((e) =>
          e.toString().contains('senha')
        )),
      );
    });
  });

  group('signUp', () {
    test('successful signup returns user', () async {
      // ... teste de sucesso
    });

    test('duplicate email throws exception', () async {
      // Arrange
      when(mockSupabase.auth.signUp(
        email: anyNamed('email'),
        password: anyNamed('password'),
      )).thenThrow(AuthException('User already registered'));

      // Act & Assert
      expect(
        () => authService.signUp('existing@email.com', 'pass123'),
        throwsA(predicate((e) =>
          e.toString().contains('já está cadastrado')
        )),
      );
    });

    test('weak password throws exception', () async {
      // ... teste de senha fraca
    });
  });
});

// Widget tests para UI
group('LoginScreen Widget', () {
  testWidgets('displays email and password fields', (tester) async {
    await tester.pumpWidget(
      MaterialApp(home: LoginScreen()),
    );

    expect(find.byType(TextField), findsNWidgets(2));
    expect(find.text('Email'), findsOneWidget);
    expect(find.text('Senha'), findsOneWidget);
  });

  testWidgets('shows error when login fails', (tester) async {
```

```
    // Mock auth service que sempre falha
    final mockAuthService = MockAuthService();
    when(mockAuthService.signIn(any, any))
        .thenThrow(Exception('Credenciais inválidas'));

    await tester.pumpWidget(
      MaterialApp(
        home: LoginScreen(authService: mockAuthService),
      ),
    );

    // Preencher campos
    await tester.enterText(
      find.byType(TextField).first,
      'test@email.com',
    );
    await tester.enterText(
      find.byType(TextField).last,
      'wrongpass',
    );

    // Clicar em login
    await tester.tap(find.text('Entrar'));
    await tester.pumpAndSettle();

    // Verificar que erro é mostrado
    expect(find.text('Credenciais inválidas'), findsOneWidget);
  });

  testWidgets('disables button during loading', (tester) async {
    // ... teste de loading state
  });
});
```

📝 Explicação

Cenários essenciais para testar:

Happy Path:

- ✅ Login com credenciais válidas
- ✅ Cadastro com dados válidos
- ✅ Recuperação de senha

Error Paths:

- ✅ Credenciais inválidas
- ✅ Email já cadastrado
- ✅ Senha fraca
- ✅ Campos vazios
- ✅ Email mal formatado
- ✅ Sem internet
- ✅ Timeout
- ✅ Servidor indisponível
- ✅ Email não confirmado

Edge Cases:

- ✅ Múltiplos cliques no botão
- ✅ Navegação durante loading
- ✅ Background/foreground durante operação
- ✅ Teclado cobre campos
- ✅ Rotação de tela

Types de testes:

1. **Unit tests** - Lógica isolada (validators, services)
2. **Widget tests** - UI components
3. **Integration tests** - Fluxos completos

Erro #11: Não Validar Email no Backend

🔴 Sintoma

- Usuários criam contas com emails fake
- Alto bounce rate em emails
- Contas spam

🔍 Causa

Confiar apenas na validação client-side.

❌ Código Errado

dart

```dart
// Apenas validação client-side
String? validateEmail(String? value) {
  if (value == null || !value.contains('@')) {
    return 'Email inválido';
  }
  return null;
}

// Supabase aceita qualquer email que passe na validação básica
```

✅ Código Correto

dart

```
// 1. Validação client-side robusta
String? validateEmail(String? value) {
  if (value == null || value.trim().isEmpty) {
    return 'Digite seu email';
  }

  // Regex mais rigoroso (mas não perfeito)
  final emailRegex = RegExp(
    r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
  );

  if (!emailRegex.hasMatch(value)) {
    return 'Digite um email válido';
  }

  // Verificar domínios comuns com typos
  final commonTypos = {
    'gmial.com': 'gmail.com',
    'gmai.com': 'gmail.com',
    'yahooo.com': 'yahoo.com',
  };

  for (final entry in commonTypos.entries) {
    if (value.endsWith(entry.key)) {
      return 'Você quis dizer ${value.replaceAll(entry.key, entry.value)}?';
    }
  }

  return null;
}

// 2. Configurar confirmação de email no Supabase
// Authentication → Settings → Enable email confirmations

// 3. Implementar reenvio de confirmação
Future<void> resendConfirmationEmail(String email) async {
  try {
    await supabase.auth.resend(
      type: OtpType.signup,
      email: email,
    );
  } catch (e) {
    throw Exception('Erro ao reenviar email: $e');
  }
}
```

```
// 4. Validação adicional com serviço externo (opcional)
// Use API como AbstractAPI, EmailListVerify, etc
Future<bool> verifyEmailExists(String email) async {
  try {
    final response = await http.get(
      Uri.parse('https://emailvalidation.abstractapi.com/v1/?api_key=$key&email=$email'),
    );

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      return data['is_valid_format']['value'] == true &&
          data['is_smtp_valid']['value'] == true;
    }

    return false;
  } catch (e) {
    // Em caso de erro, permitir (não bloquear usuário)
    return true;
  }
}

// 5. Implementar rate limiting no backend
// Use Supabase Edge Functions com Upstash Rate Limiting
```

📝 Explicação

Camadas de validação:

1. **Client-side** - Feedback instantâneo, UX
2. **Email confirmation** - Verifica que email existe
3. **Rate limiting** - Previne spam
4. **External validation** (opcional) - Verifica email real

No Supabase Dashboard:

- Habilite "Enable email confirmations"
- Configure template de email
- Ajuste "Minimum password length"

Erro #12: Não Implementar Rate Limiting

🔴 Sintoma

- Ataques de força bruta
- Spam de cadastros
- Custos elevados de API

🔍 Causa

Permitir tentativas ilimitadas de login/cadastro.

❌ Código Errado

dart

```dart
// Sem limite de tentativas
Future<void> _handleLogin() async {
  try {
    await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );
  } catch (e) {
    // Pode tentar infinitamente
    showError(e.toString());
  }
}
```

✅ Código Correto

dart

```dart
// 1. Rate limiting client-side (básico)
class LoginScreen extends StatefulWidget {
  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  int _loginAttempts = 0;
  DateTime? _lastAttempt;
  static const _maxAttempts = 5;
  static const _lockoutDuration = Duration(minutes: 15);

  bool get _isLockedOut {
    if (_lastAttempt == null) return false;

    if (_loginAttempts >= _maxAttempts) {
      final timeSinceLastAttempt = DateTime.now().difference(_lastAttempt!);
      return timeSinceLastAttempt < _lockoutDuration;
    }

    return false;
  }

  Duration get _remainingLockoutTime {
    if (_lastAttempt == null) return Duration.zero;

    final elapsed = DateTime.now().difference(_lastAttempt!);
    final remaining = _lockoutDuration - elapsed;

    return remaining.isNegative ? Duration.zero : remaining;
  }

  Future<void> _handleLogin() async {
    // Verificar se está bloqueado
    if (_isLockedOut) {
      final minutes = _remainingLockoutTime.inMinutes;
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            'Muitas tentativas falhas. Tente novamente em $minutes minutos.',
          ),
          backgroundColor: Colors.red,
        ),
      );
      return;
    }
```

```
    try {
      await supabase.auth.signInWithPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text,
      );

      // Reset tentativas em caso de sucesso
      setState(() {
        _loginAttempts = 0;
        _lastAttempt = null;
      });

      if (!mounted) return;
      Navigator.pushReplacement(context, ...);

    } on AuthException catch (e) {
      // Incrementar tentativas
      setState(() {
        _loginAttempts++;
        _lastAttempt = DateTime.now();
      });

      if (!mounted) return;

      // Mensagem baseada em tentativas
      final attemptsLeft = _maxAttempts - _loginAttempts;

      String message;
      if (_isLockedOut) {
        message = 'Conta temporariamente bloqueada por segurança.';
      } else if (attemptsLeft <= 2) {
        message = 'Credenciais incorretas. $attemptsLeft tentativas restantes.';
      } else {
        message = 'Email ou senha incorretos.';
      }

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(message),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  @override
```

```
  Widget build(BuildContext context) {
    return Column(
      children: [
        // ... campos ...

        ElevatedButton(
          onPressed: _isLockedOut ? null : _handleLogin,
          child: Text(
            _isLockedOut
                ? 'Bloqueado (${_remainingLockoutTime.inMinutes}min)'
                : 'Entrar',
          ),
        ),
      ],
    );
  }
}

// 2. Rate limiting server-side (Supabase Edge Function)
// supabase/functions/rate-limit-login/index.ts
import { serve } from 'https://deno.land/std@0.168.0/http/server.ts'
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

const rateLimit = new Map<string, { count: number; timestamp: number }>()

serve(async (req) => {
  const { email } = await req.json()
  const clientIP = req.headers.get('x-forwarded-for') || 'unknown'
  const key = `${clientIP}:${email}`

  const now = Date.now()
  const windowMs = 15 * 60 * 1000 // 15 minutos

  // Limpar entradas antigas
  for (const [k, v] of rateLimit.entries()) {
    if (now - v.timestamp > windowMs) {
      rateLimit.delete(k)
    }
  }

  // Verificar rate limit
  const current = rateLimit.get(key)

  if (current && current.count >= 5) {
    return new Response(
      JSON.stringify({ error: 'Too many attempts' }),
      { status: 429 }
```

```
    )
  }

  // Incrementar contador
  rateLimit.set(key, {
    count: current ? current.count + 1 : 1,
    timestamp: now,
  })

  return new Response(
    JSON.stringify({ allowed: true }),
    { status: 200 }
  )
})

// 3. No Supabase Dashboard
// Authentication → Rate Limits
// Configure limites apropriados
```

📝 **Explicação**

Estratégias de rate limiting:

1. **Client-side** - Proteção básica, fácil de bypass
2. **Server-side** - Proteção real, mais complexo
3. **Híbrido** - Melhor UX + segurança

Supabase oferece:

- Rate limiting automático (configurável)
- Por IP, por usuário, por endpoint
- Configurável no dashboard

Erro #13: Ignorar Acessibilidade

🔴 **Sintoma**

- Usuários com deficiência não conseguem usar o app
- Leitores de tela não funcionam corretamente
- Contraste ruim

🔍 Causa

Não pensar em acessibilidade desde o início.

❌ Código Errado

dart

```dart
// Sem semântica, sem labels, sem contraste
TextField(
  decoration: InputDecoration(
    hintText: 'Digite aqui', // Vago
  ),
)

// Botão sem label descritivo
IconButton(
  icon: Icon(Icons.login),
  onPressed: () {},
)

// Cores com baixo contraste
Text(
  'Erro',
  style: TextStyle(color: Colors.grey), // Difícil de ler
)
```

✅ Código Correto

dart

```dart
// 1. Campos com semântica apropriada
TextField(
  controller: _emailController,
  decoration: InputDecoration(
    labelText: 'Email', // Label claro
    hintText: 'seu@email.com', // Exemplo específico
    prefixIcon: Icon(Icons.email),
  ),
  keyboardType: TextInputType.emailAddress,
  textInputAction: TextInputAction.next,

  // Semântica para leitores de tela
  semanticsLabel: 'Campo de email',
)

// 2. Botões com labels descritivos
Semantics(
  label: 'Fazer login',
  button: true,
  child: ElevatedButton(
    onPressed: _handleLogin,
    child: const Text('Entrar'),
  ),
)

// Para IconButton, use Semantics ou tooltip
IconButton(
  icon: const Icon(Icons.visibility),
  tooltip: 'Mostrar senha',
  onPressed: () {
    setState(() => _obscurePassword = !_obscurePassword);
  },
)

// 3. Contraste adequado (WCAG AA mínimo)
// Use ferramentas como https://webaim.org/resources/contrastchecker/

// Erro - contraste apropriado
Container(
  padding: const EdgeInsets.all(8),
  decoration: BoxDecoration(
    color: Theme.of(context).colorScheme.errorContainer,
    borderRadius: BorderRadius.circular(8),
  ),
  child: Text(
    'Email ou senha incorretos',
```

```
    style: TextStyle(
      color: Theme.of(context).colorScheme.onErrorContainer,
      // Garantir contraste adequado
    ),
  ),
)

// 4. Tamanhos de toque adequados (mínimo 48x48)
SizedBox(
  width: double.infinity,
  height: 48, // Altura mínima recomendada
  child: ElevatedButton(
    onPressed: _handleLogin,
    child: const Text('Entrar'),
  ),
)

// 5. Focus management
FocusScope.of(context).requestFocus(_passwordFocusNode);

// 6. Anunciar mudanças importantes
void _announceError(String message) {
  SemanticsService.announce(
    message,
    TextDirection.ltr,
  );
}

// 7. Labels em formulários
Form(
  child: Column(
    children: [
      // Label + Campo agrupados semanticamente
      MergeSemantics(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const Text('Email'),
            TextField(
              controller: _emailController,
              decoration: const InputDecoration(
                hintText: 'seu@email.com',
              ),
            ),
          ],
        ),
      ),
```

```
    ],
  ),
)

// 8. Testar com TalkBack/VoiceOver
// Android: Settings → Accessibility → TalkBack
// iOS: Settings → Accessibility → VoiceOver
```

📝 Explicação

Checklist de acessibilidade:

- ☐ Semântica em todos os widgets interativos
- ☐ Labels descritivos (não "clique aqui")
- ☐ Contraste WCAG AA (mínimo 4.5:1 para texto normal)
- ☐ Tamanhos de toque >= 48x48
- ☐ Focus navigation funciona
- ☐ Testado com leitor de tela
- ☐ Erros são anunciados
- ☐ Suporte a fontes maiores (acessibility text scale)

Erro #14: Não Tratar Estado "mounted"

🔴 Sintoma

- Erro: "setState() called after dispose()"
- App trava ao navegar rapidamente
- Warnings no console

🔍 Causa

Chamar `setState()` após widget ser removido da árvore.

❌ Código Errado

dart

```dart
Future<void> _handleLogin() async {
  try {
    await supabase.auth.signInWithPassword(
      email: email,
      password: password,
    );

    // ❌ Pode falhar se usuário já navegou para outra tela
    Navigator.pushReplacement(context, ...);
    setState(() => _isLoading = false);

  } catch (e) {
    // ❌ Widget pode ter sido disposed
    setState(() => _isLoading = false);
    ScaffoldMessenger.of(context).showSnackBar(...);
  }
}
```

✅ Código Correto

dart

```dart
Future<void> _handleLogin() async {
  setState(() => _isLoading = true);

  try {
    await supabase.auth.signInWithPassword(
      email: _emailController.text.trim(),
      password: _passwordController.text,
    );

    // ✅ Verificar se widget ainda está montado
    if (!mounted) return;

    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => const HomeScreen()),
    );

  } catch (e) {
    // ✅ Verificar antes de setState
    if (!mounted) return;

    setState(() => _isLoading = false);

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

// Padrão completo com finally
Future<void> _handleLoginComplete() async {
  setState(() => _isLoading = true);

  try {
    await supabase.auth.signInWithPassword(
      email: _emailController.text.trim(),
      password: _passwordController.text,
    );

    if (!mounted) return;
    Navigator.pushReplacement(context, ...);

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

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

```
      SnackBar(content: Text(e.toString())),
    );
  } finally {
    // ✅ Verificar antes de setState no finally também
    if (mounted) {
      setState(() => _isLoading = false);
    }
  }
}
```

📝 Explicação

Quando verificar `mounted`:

- ✅ Antes de qualquer `setState()`
- ✅ Antes de usar `context` (Navigator, Theme, MediaQuery, etc)
- ✅ Após qualquer operação assíncrona (await)
- ✅ Em callbacks de timers ou streams

Pattern:

```dart
// Sempre:
if (!mounted) return;
setState(() => ...);

// Ou:
if (mounted) {
  setState(() => ...);
}
```

Erro #15: Hardcoded Strings e Magic Numbers

🔴 Sintoma

- Difícil manter textos consistentes
- Impossível internacionalizar depois
- Números "mágicos" sem significado

🔍 Causa

Strings e valores numéricos espalhados pelo código.

❌ Código Errado

dart

```dart
// Strings hardcoded
Text('Digite seu email'),
Text('Email inválido'), // Repetido em vários lugares
Text('Erro ao fazer login'), // Inconsistente

// Magic numbers
if (password.length < 6) // Por que 6?
SizedBox(height: 16) // Por que 16?
Duration(seconds: 15) // Por que 15?
```

✅ Código Correto

dart

```dart
// 1. Criar arquivo de constantes
// lib/core/constants/app_strings.dart
class AppStrings {
  // Títulos
  static const String appName = 'MeuApp';
  static const String loginTitle = 'Bem-vindo de volta';
  static const String signupTitle = 'Criar Conta';

  // Labels
  static const String emailLabel = 'Email';
  static const String passwordLabel = 'Senha';
  static const String confirmPasswordLabel = 'Confirmar Senha';

  // Hints
  static const String emailHint = 'seu@email.com';
  static const String passwordHint = '••••••••';

  // Botões
  static const String loginButton = 'Entrar';
  static const String signupButton = 'Criar Conta';
  static const String logoutButton = 'Sair';

  // Erros
  static const String emailEmptyError = 'Digite seu email';
  static const String emailInvalidError = 'Email inválido';
  static const String passwordEmptyError = 'Digite sua senha';
  static const String passwordWeakError =
      'Senha deve ter no mínimo ${AppConstants.minPasswordLength} caracteres';
  static const String passwordMismatchError = 'As senhas não correspondem';
  static const String loginFailedError = 'Email ou senha incorretos';
  static const String networkError =
      'Sem conexão. Verifique sua internet e tente novamente.';

  // Mensagens de sucesso
  static const String loginSuccess = 'Login realizado com sucesso!';
  static const String signupSuccess = 'Conta criada! Verifique seu email.';
}

// lib/core/constants/app_constants.dart
class AppConstants {
  // Validação
  static const int minPasswordLength = 6;
  static const int maxPasswordLength = 128;
  static const int minNameLength = 2;

  // Timeouts
```

```dart
  static const Duration apiTimeout = Duration(seconds: 15);
  static const Duration debounceDelay = Duration(milliseconds: 500);

  // Rate limiting
  static const int maxLoginAttempts = 5;
  static const Duration lockoutDuration = Duration(minutes: 15);

  // UI
  static const double buttonHeight = 48.0;
  static const double borderRadius = 12.0;
  static const double defaultPadding = 16.0;
  static const double largePadding = 24.0;
}

// lib/core/constants/app_colors.dart
class AppColors {
  // Light theme
  static const Color primaryLight = Color(0xFF6750A4);
  static const Color errorLight = Color(0xFFBA1A1A);
  // ... outras cores
}

// 2. Usar as constantes
class LoginScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(AppStrings.loginTitle), // ✅
      ),
      body: Padding(
        padding: const EdgeInsets.all(AppConstants.defaultPadding), // ✅
        child: Column(
          children: [
            TextFormField(
              decoration: InputDecoration(
                labelText: AppStrings.emailLabel, // ✅
                hintText: AppStrings.emailHint, // ✅
              ),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return AppStrings.emailEmptyError; // ✅
                }
                return null;
              },
            ),
```

```
          SizedBox(height: AppConstants.defaultPadding), // ✅

          SizedBox(
            width: double.infinity,
            height: AppConstants.buttonHeight, // ✅
            child: ElevatedButton(
              onPressed: _handleLogin,
              child: const Text(AppStrings.loginButton), // ✅
            ),
          ),
        ],
      ),
    ),
  );
 }
}

// 3. Para internacionalização (i18n)
// Use o pacote flutter_localizations

// lib/l10n/app_en.arb
{
 "loginTitle": "Welcome Back",
 "emailLabel": "Email",
 "loginButton": "Login"
}

// lib/l10n/app_pt.arb
{
 "loginTitle": "Bem-vindo de volta",
 "emailLabel": "Email",
 "loginButton": "Entrar"
}

// Usar no código
Text(AppLocalizations.of(context).loginTitle)
```

📝 Explicação

Benefícios:

- ✅ Fácil manter consistência
- ✅ Preparado para i18n
- ✅ Código mais legível
- ✅ Fácil atualizar todos os textos

- ✅ Previne typos

Estrutura recomendada:

```
lib/
└── core/
    └── constants/
        ├── app_strings.dart
        ├── app_constants.dart
        ├── app_colors.dart
        └── app_assets.dart
```

Bônus: Ferramentas de Debug

Ferramentas Úteis

dart

```dart
// 1. Logger configurado
final logger = Logger(
  printer: PrettyPrinter(),
  level: kDebugMode ? Level.debug : Level.error,
);

// 2. DevTools Supabase
// Acesse o dashboard em tempo real
// https://app.supabase.com/project/_/auth/users

// 3. Flutter DevTools
// flutter pub global activate devtools
// flutter pub global run devtools

// 4. Analyze & Lint
// flutter analyze
// dart fix --apply

// 5. Custom error widget
void main() {
  ErrorWidget.builder = (FlutterErrorDetails details) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text(
            kDebugMode ? details.exception.toString() : 'Erro no app',
          ),
        ),
      ),
    );
  };

  runApp(const MyApp());
}
```

Conclusão da Seção

Você agora conhece os 15 erros mais comuns em autenticação Flutter e como evitá-los:

1. ✅ Não validar antes de enviar
2. ✅ Não tratar erros de rede
3. ✅ Armazenar dados sensíveis inseguramente

4. ✅ Não dar feedback visual
5. ✅ Não limpar controllers
6. ✅ Deep linking não funciona
7. ✅ Expor informações em logs
8. ✅ Não lidar com sessões expiradas
9. ✅ URLs incorretas
10. ✅ Não testar cenários de erro
11. ✅ Não validar email no backend
12. ✅ Não implementar rate limiting
13. ✅ Ignorar acessibilidade
14. ✅ Não tratar estado `mounted`
15. ✅ Hardcoded strings e magic numbers

Práticas finais:

- 📖 Leia este guia antes de começar
- ✅ Use como checklist durante desenvolvimento
- 🔍 Revise código com estes pontos em mente
- 🧪 Teste cenários de erro, não apenas happy path
- 📝 Documente decisões e configurações

Recursos adicionais:

- Flutter Best Practices
- Supabase Security Best Practices
- OWASP Mobile Security

🎉 Parabéns!

Você completou todas as 14 seções do material didático sobre **Autenticação em Flutter com Supabase**!

Agora você domina:

- ✅ Fundamentos teóricos
- ✅ Arquitetura e organização

- ✅ UI/UX e acessibilidade
- ✅ Segurança e validações
- ✅ Gerenciamento de estado
- ✅ Integração com APIs
- ✅ Supabase completo
- ✅ Design profissional
- ✅ Desafios práticos
- ✅ Troubleshooting

Próximos passos:

1. Complete um dos desafios da Seção 13
2. Revise os erros comuns desta seção periodicamente
3. Contribua com a comunidade Flutter!

Boa sorte em seus projetos! 🚀

Fim da Seção 14: Erros Comuns e Como Evitá-los Fim do Material Didático Completo