As Telas do Fluxo de Autenticação

Desenvolvimento de Aplicativos Móveis - Flutter

Introdução

Agora que compreendemos os fundamentos teóricos e estabelecemos uma arquitetura sólida, vamos detalhar cada tela que compõe o fluxo completo de autenticação. Estas telas são a face do seu sistema de autenticação, o ponto de contato direto com o usuário. Precisamos garantir que cada uma seja intuitiva, funcional e transmita confiança.

Vamos explorar não apenas o que cada tela deve conter, mas também por que cada elemento está lá e como eles trabalham juntos para criar uma experiência coesa. Cada decisão de design que fazemos nessas telas impacta diretamente como o usuário percebe e interage com nossa aplicação.

A Jornada Completa do Usuário

Antes de mergulharmos em cada tela individualmente, é importante visualizar a jornada completa que um usuário percorre desde o primeiro contato com o aplicativo até estar autenticado e usando suas funcionalidades principais.

Quando um usuário novo abre seu aplicativo pela primeira vez, ele encontra uma tela inicial que rapidamente determina que não há sessão ativa e o redireciona para a tela de login. Esta tela de login é sua oportunidade de causar uma primeira impressão positiva. Ela deve ser limpa, convidativa e deixar claro que o processo será rápido e simples.

Se o usuário ainda não tem conta, ele clica em um link que o leva para a tela de cadastro. Aqui ele fornece informações básicas e cria suas credenciais. Após cadastro bem-sucedido, dependendo de como você configurou, ele pode ser logado automaticamente ou precisar confirmar seu email primeiro. Se esqueceu a senha, há um caminho claro para recuperação que envolve a tela de recuperação de senha e posteriormente uma tela para definir nova senha.

Usuários que retornam ao aplicativo têm uma experiência diferente. A tela inicial detecta que há uma sessão válida salva e os leva diretamente para a área autenticada do app, sem necessidade de fazer login novamente. Esta persistência de sessão é crucial para uma boa experiência em aplicativos móveis, onde usuários esperam permanecer logados entre usos.

Compreender este fluxo completo nos ajuda a desenhar cada tela individual de forma que se conecte naturalmente às outras, criando uma jornada coesa ao invés de telas isoladas.

Tela de Login: A Porta de Entrada

A tela de login é frequentemente o primeiro contato significativo do usuário com seu aplicativo. Ela precisa ser limpa, focada e transmitir profissionalismo e confiança. Vamos dissecar cada elemento desta tela e entender por que ele está lá.

Elementos Visuais Essenciais

No topo da tela, reserve espaço para o logo ou nome do seu aplicativo. Este elemento de branding é importante porque confirma ao usuário que ele está no lugar certo. Posicione o logo centralizado com espaçamento generoso ao redor para dar respiro visual. O logo não precisa ser enorme, ele é um elemento de orientação, não o foco principal da tela.

Logo abaixo do logo, coloque um título de boas-vindas. Algo como "Bem-vindo de volta" ou simplesmente "Login" funciona bem. Este título serve como marcador visual que orienta o usuário sobre o que esta tela faz. Use tipografia grande e bold para este título, tornando-o claramente legível em uma rápida olhada.

O corpo principal da tela contém os campos de entrada. Para login tradicional, você precisa de dois campos: email e senha. Cada campo deve seguir as melhores práticas de design de formulários móveis. Use TextField com estilo filled (preenchido com cor de fundo sutil) ao invés de outlined (apenas borda). O estilo filled é mais moderno e funciona melhor em interfaces móveis porque a área clicável é mais óbvia.

Configuração dos Campos de Texto

O campo de email deve ter algumas configurações específicas que melhoram dramaticamente a experiência do usuário. Primeiro, defina o tipo de teclado como emailAddress, o que faz o sistema operacional mostrar um teclado otimizado com o símbolo arroba facilmente acessível e sugestões de domínios comuns. Segundo, desabilite autocapitalização, pois emails são sempre minúsculos. Terceiro, desabilite autocorreção, que só atrapalha ao digitar endereços de email.

Aqui está como configurar o campo de email corretamente:

```dart
```

```dart
TextField(
  controller: _emailController,
  keyboardType: TextInputType.emailAddress,
  textInputAction: TextInputAction.next,
  autocorrect: false,
  enableSuggestions: false,
  textCapitalization: TextCapitalization.none,
  decoration: InputDecoration(
    labelText: 'Email',
    hintText: 'seu@email.com',
    prefixIcon: Icon(Icons.email_outlined),
  ),
  onSubmitted: (_) {
    // Move foco para o próximo campo quando usuário pressiona "Next"
    FocusScope.of(context).nextFocus();
  },
)
```

Note como estamos sendo específicos sobre cada aspecto do comportamento do campo. O textInputAction define que o botão do teclado mostra "Next" ao invés de "Done", e o callback onSubmitted garante que pressionar esse botão realmente move o foco para o próximo campo. Estes pequenos detalhes fazem o aplicativo parecer polido e bem pensado.

O campo de senha requer considerações adicionais. Por padrão, caracteres devem ser obscurecidos por segurança, mas você deve oferecer ao usuário a opção de revelar temporariamente o que está digitando. Muitos erros de login acontecem simplesmente porque o usuário digitou a senha errada mas não conseguiu ver para corrigir.

Implemente um ícone de olho no final do campo que permite alternar entre texto visível e obscurecido:

dart

```
class _LoginPageState extends State<LoginPage> {
  bool _obscurePassword = true;

  @override
  Widget build(BuildContext context) {
    return TextField(
      controller: _passwordController,
      obscureText: _obscurePassword,
      textInputAction: TextInputAction.done,
      decoration: InputDecoration(
        labelText: 'Senha',
        prefixIcon: Icon(Icons.lock_outlined),
        suffixIcon: IconButton(
          icon: Icon(
            _obscurePassword
              ? Icons.visibility_outlined
              : Icons.visibility_off_outlined,
          ),
          onPressed: () {
            setState(() {
              _obscurePassword = !_obscurePassword;
            });
          },
        ),
      ),
      onSubmitted: (_) {
        // Quando usuário pressiona "Done" no teclado, tenta fazer login
        _handleLogin();
      },
    );
  }
}
```

Aqui mantemos um estado local obscurePassword que controla se o texto está visível. O ícone muda entre olho aberto e fechado baseado neste estado, dando feedback visual claro. Quando o usuário pressiona "Done" no teclado do campo de senha, tentamos fazer login automaticamente, economizando um toque adicional.

Elementos de Navegação e Ação

Entre os campos e o botão principal de login, coloque um link "Esqueceu a senha?" alinhado à direita. Este link deve ser discreto mas clicável, tipicamente em uma cor secundária e tamanho de fonte ligeiramente menor que o texto dos campos. Não force o usuário a procurar por esta opção se precisar dela.

O botão de login é o elemento de ação primário da tela e deve ter destaque visual correspondente. Use um botão filled com a cor primária da sua aplicação, largo o suficiente para preencher boa parte da largura da tela

(deixando margens laterais de dezesseis a vinte e quatro pixels), e com altura generosa para facilitar o toque. O texto do botão deve ser grande, em cor contrastante, e direto ao ponto: "Entrar" ou "Fazer Login".

Durante operações assíncronas, o botão deve dar feedback visual imediato. Substitua o texto por um indicador de progresso circular e desabilite o botão para prevenir múltiplos toques. Esta transformação do botão comunica claramente ao usuário que sua ação foi registrada e está sendo processada:

```dart
ElevatedButton(
  onPressed: _isLoading ? null : _handleLogin,
  style: ElevatedButton.styleFrom(
    minimumSize: Size(double.infinity, 56),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(12),
    ),
  ),
  child: _isLoading
    ? SizedBox(
        height: 20,
        width: 20,
        child: CircularProgressIndicator(
          strokeWidth: 2,
          valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
        ),
      )
    : Text(
        'Entrar',
        style: TextStyle(
          fontSize: 16,
          fontWeight: FontWeight.w600,
        ),
      ),
)
```

Opções Alternativas de Login

Se você implementou métodos alternativos de autenticação como login social, eles devem aparecer após um divisor visual que separa claramente as opções. Use um divisor horizontal com texto centralizado como "ou continue com" para deixar explícito que são alternativas ao login tradicional, não passos adicionais.

Os botões de login social têm convenções visuais específicas que você deve seguir. O botão do Google tipicamente tem fundo branco com borda sutil e o logo do Google à esquerda. O botão da Apple tem fundo preto com logo branco. Seguir estas convenções visuais ajuda usuários a reconhecer instantaneamente as opções e criar confiança, pois eles estão familiarizados com esses padrões de outras aplicações.

Link para Cadastro

No rodapé da tela, após todos os outros elementos, coloque um texto explicativo seguido de um link para cadastro. Algo como "Não tem conta? Cadastre-se" funciona bem. O texto explicativo deve estar em cor neutra, enquanto "Cadastre-se" deve ser um link clicável destacado. Este elemento não deve competir visualmente com o botão de login, mas precisa ser facilmente encontrável por usuários novos.

Tela de Cadastro: Criando Novas Contas

A tela de cadastro é onde você conquista novos usuários, mas também onde corre o risco de perdê-los se o processo for muito complicado ou demorado. O princípio fundamental aqui é coletar apenas o mínimo necessário. Cada campo adicional aumenta a fricção e a chance do usuário desistir.

Campos Essenciais

Para um cadastro básico, você precisa de três campos obrigatórios: email, senha e confirmação de senha. Considere adicionar um campo de nome se você precisará personalizar a experiência do usuário, mas seja criterioso. Outros dados podem ser coletados posteriormente, após o usuário já estar engajado com o aplicativo.

O campo de email tem as mesmas configurações que discutimos para a tela de login. O campo de senha, porém, tem requisitos adicionais na tela de cadastro porque você está definindo uma nova senha e precisa garantir que seja forte o suficiente.

Indicadores de Força de Senha

Ao invés de apenas rejeitar senhas fracas ao tentar enviar, transforme a validação em um guia interativo. Mostre os requisitos de senha próximos ao campo e indique visualmente quais foram atendidos à medida que o usuário digita. Isto transforma validação em orientação proativa, uma experiência muito melhor que descobrir erros apenas ao final.

Aqui está uma implementação de indicador visual de requisitos:

dart

```dart
class PasswordRequirements extends StatelessWidget {
  final String password;

  const PasswordRequirements({Key? key, required this.password})
    : super(key: key);

  bool get hasMinLength => password.length >= 8;
  bool get hasLetter => password.contains(RegExp(r'[a-zA-Z]'));
  bool get hasNumber => password.contains(RegExp(r'[0-9]'));

  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        _RequirementItem(
          text: 'Mínimo 8 caracteres',
          isMet: hasMinLength,
        ),
        SizedBox(height: 4),
        _RequirementItem(
          text: 'Pelo menos uma letra',
          isMet: hasLetter,
        ),
        SizedBox(height: 4),
        _RequirementItem(
          text: 'Pelo menos um número',
          isMet: hasNumber,
        ),
      ],
    );
  }
}

class _RequirementItem extends StatelessWidget {
  final String text;
  final bool isMet;

  const _RequirementItem({
    required this.text,
    required this.isMet,
  });

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

```
      children: [
        Icon(
          isMet ? Icons.check_circle : Icons.circle_outlined,
          size: 16,
          color: isMet
              ? Colors.green
              : Theme.of(context).colorScheme.outline,
        ),
        SizedBox(width: 8),
        Text(
          text,
          style: TextStyle(
            fontSize: 12,
            color: isMet
                ? Colors.green
                : Theme.of(context).colorScheme.onSurfaceVariant,
          ),
        ),
      ],
    );
  }
}
```

Este widget mostra cada requisito com um ícone que muda de um círculo vazio para um check verde à medida que o requisito é atendido. O texto também muda de cor para reforçar visualmente. Posicione este widget logo abaixo do campo de senha, atualizando em tempo real conforme o usuário digita.

Campo de Confirmação de Senha

O campo de confirmação de senha existe para capturar erros de digitação. A validação aqui é simples: o valor deve ser idêntico ao campo de senha. Porém, o timing desta validação importa. Se você mostrar erro antes do usuário terminar de digitar, é frustrante. Se mostrar apenas ao submeter o formulário, o usuário pode precisar corrigir e tentar novamente.

Uma boa abordagem intermediária é validar quando o usuário sai do campo de confirmação. Se os valores não coincidem, mostre erro imediatamente para que ele possa corrigir antes de tentar submeter. Se coincidem, você pode mostrar um checkmark verde de confirmação visual:

```
dart
```

```dart
TextFormField(
  controller: _confirmPasswordController,
  obscureText: true,
  decoration: InputDecoration(
    labelText: 'Confirmar senha',
    prefixIcon: Icon(Icons.lock_outlined),
    suffixIcon: _confirmPasswordController.text.isNotEmpty &&
            _confirmPasswordController.text == _passwordController.text
      ? Icon(Icons.check_circle, color: Colors.green)
      : null,
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Por favor, confirme sua senha';
    }
    if (value != _passwordController.text) {
      return 'As senhas não coincidem';
    }
    return null;
  },
  onChanged: (_) {
    // Força reconstrução para atualizar ícone de check
    setState(() {});
  },
)
```

Termos de Uso e Política de Privacidade

É importante, tanto do ponto de vista legal quanto ético, que usuários saibam e concordem com como seus dados serão usados. Inclua um checkbox onde o usuário confirma que leu e aceita os termos de uso e política de privacidade. Os termos devem ser links clicáveis que levam para documentos completos.

Não use checkboxes pré-marcados. O usuário deve tomar a ação consciente de marcar o checkbox. Alguns desenvolvedores tentam esconder isso ou tornar automático, mas além de poder ser ilegal em muitas jurisdições, é eticamente questionável. Seja transparente e respeite a autonomia do usuário:

```dart
```

```
Row(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
  Checkbox(
   value: _acceptedTerms,
   onChanged: (value) {
    setState(() {
     _acceptedTerms = value ?? false;
    });
   },
  ),
  Expanded(
   child: GestureDetector(
    onTap: () {
     setState(() {
      _acceptedTerms = !_acceptedTerms;
     });
    },
    child: RichText(
     text: TextSpan(
      style: TextStyle(
       fontSize: 12,
       color: Theme.of(context).colorScheme.onSurface,
      ),
      children: [
       TextSpan(text: 'Aceito os '),
       TextSpan(
        text: 'termos de uso',
        style: TextStyle(
         color: Theme.of(context).colorScheme.primary,
         decoration: TextDecoration.underline,
        ),
        recognizer: TapGestureRecognizer()
         ..onTap = () {
          // Abrir tela ou URL com termos de uso
         },
       ),
       TextSpan(text: ' e '),
       TextSpan(
        text: 'política de privacidade',
        style: TextStyle(
         color: Theme.of(context).colorScheme.primary,
         decoration: TextDecoration.underline,
        ),
        recognizer: TapGestureRecognizer()
         ..onTap = () {
```

```
                // Abrir tela ou URL com política de privacidade
              },
            ),
          ],
        ),
      ),
    ),
  ),
 ],
)
```

Botão de Cadastro e Feedback

O botão de cadastro deve ser desabilitado até que todos os campos estejam válidos e o checkbox de termos esteja marcado. Isto previne frustrações de tentar submeter e receber erros de validação. O usuário pode ver que o botão está desabilitado e olhar o formulário para identificar o que falta.

Após cadastro bem-sucedido, você tem algumas opções de fluxo. Se não exige confirmação de email, pode logar o usuário automaticamente e levá-lo para dentro do aplicativo. Se exige confirmação, redirecione para uma tela explicativa informando que um email foi enviado e instruindo o usuário a verificar sua caixa de entrada. Esta tela intermediária é importante porque sem ela o usuário pode ficar confuso sobre por que não foi logado automaticamente.

Tela de Recuperação de Senha: Ajudando Usuários Perdidos

Usuários esquecem senhas. É inevitável e não devemos tornar este processo doloroso. A tela de recuperação de senha deve ser simples e tranquilizadora, comunicando claramente que resolver o problema é fácil e rápido.

Interface Minimalista

Esta tela é intencionalmente simples porque tem apenas uma tarefa. No topo, logo e título "Recuperar senha" estabelecem contexto. Logo abaixo, um parágrafo explicativo curto e direto: "Digite seu email e enviaremos instruções para redefinir sua senha". Esta explicação antecipa a pergunta do usuário antes que ele precise fazê-la.

O corpo da tela contém apenas um campo de email e um botão para enviar. Não peça informações adicionais. Quanto mais simples o processo, menor a fricção e maior a taxa de completação. O campo de email deve ter as mesmas configurações que usamos nas outras telas, com teclado otimizado e validação de formato.

Estados da Tela

Esta tela tem dois estados distintos que representam diferentes momentos do fluxo. O primeiro estado é o formulário inicial onde o usuário digita o email. O segundo estado, após envio bem-sucedido, é uma mensagem de confirmação. Transformar a tela ao invés de navegar para outra mantém o contexto e é menos confuso.

No estado de confirmação, substitua o formulário por uma mensagem de sucesso clara. Use um ícone grande de email ou check para reforço visual. O texto deve confirmar que o email foi enviado e dar instruções específicas: "Verifique sua caixa de entrada e pasta de spam. O link de recuperação expira em uma hora". Esta especificidade ajuda o usuário saber exatamente o que fazer e quando.

dart

```dart
class ForgotPasswordPage extends StatefulWidget {
  @override
  _ForgotPasswordPageState createState() => _ForgotPasswordPageState();
}

class _ForgotPasswordPageState extends State<ForgotPasswordPage> {
  final _emailController = TextEditingController();
  bool _emailSent = false;
  bool _isLoading = false;

  Future<void> _sendResetEmail() async {
    setState(() => _isLoading = true);

    try {
      await authService.resetPassword(_emailController.text);
      setState(() {
        _emailSent = true;
        _isLoading = false;
      });
    } catch (e) {
      setState(() => _isLoading = false);
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Erro ao enviar email: $e')),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Recuperar senha'),
      ),
      body: Padding(
        padding: EdgeInsets.all(24),
        child: _emailSent ? _buildSuccessView() : _buildFormView(),
      ),
    );
  }

  Widget _buildFormView() {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        SizedBox(height: 24),
        Text(
```

```
        'Digite seu email e enviaremos instruções para redefinir sua senha',
        style: Theme.of(context).textTheme.bodyLarge,
        textAlign: TextAlign.center,
      ),
      SizedBox(height: 32),
      TextField(
        controller: _emailController,
        keyboardType: TextInputType.emailAddress,
        decoration: InputDecoration(
          labelText: 'Email',
          prefixIcon: Icon(Icons.email_outlined),
        ),
      ),
      SizedBox(height: 24),
      ElevatedButton(
        onPressed: _isLoading ? null : _sendResetEmail,
        child: _isLoading
          ? CircularProgressIndicator()
          : Text('Enviar link de recuperação'),
      ),
    ],
  );
}

Widget _buildSuccessView() {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Icon(
        Icons.mark_email_read_outlined,
        size: 80,
        color: Theme.of(context).colorScheme.primary,
      ),
      SizedBox(height: 24),
      Text(
        'Email enviado!',
        style: Theme.of(context).textTheme.headlineMedium,
      ),
      SizedBox(height: 16),
      Text(
        'Verifique sua caixa de entrada em ${_emailController.text}',
        textAlign: TextAlign.center,
        style: Theme.of(context).textTheme.bodyLarge,
      ),
      SizedBox(height: 8),
      Text(
        'O link de recuperação expira em 1 hora',
```

```
          textAlign: TextAlign.center,
          style: Theme.of(context).textTheme.bodySmall?.copyWith(
            color: Theme.of(context).colorScheme.onSurfaceVariant,
          ),
        ),
        SizedBox(height: 32),
        OutlinedButton(
          onPressed: _canResend ? _sendResetEmail : null,
          child: Text('Reenviar email'),
        ),
      ],
    );
  }
}
```

Proteção Contra Abuso

Um aspecto importante é prevenir que usuários mal-intencionados abusem da funcionalidade de recuperação enviando spam de emails. Implemente um cooldown que permite reenvio apenas após um período mínimo, tipicamente sessenta segundos. Durante este período, o botão de reenvio fica desabilitado e você pode mostrar um contador regressivo para informar quando ficará disponível novamente.

Tela de Redefinição de Senha: Completando a Recuperação

Quando o usuário clica no link do email de recuperação, ele chega nesta tela onde pode definir uma nova senha. Esta tela é acessada através de deep linking, um aspecto técnico que configuraremos mais adiante, mas por enquanto vamos focar na interface e experiência.

Campos e Validações

A tela contém dois campos: nova senha e confirmação da nova senha. As mesmas validações e indicadores visuais de força que usamos na tela de cadastro devem ser aplicados aqui. O usuário está criando uma nova senha, então queremos garantir que seja forte.

Mostre os requisitos de senha com checkmarks que atualizam em tempo real, exatamente como na tela de cadastro. Esta consistência visual ajuda o usuário a reconhecer o padrão e entender imediatamente o que precisa fazer.

Feedback de Sucesso

Após redefinir a senha com sucesso, mostre uma mensagem clara de confirmação antes de redirecionar. Algo como "Senha atualizada com sucesso! Você será redirecionado para o login". Esta mensagem interim é importante porque confirma que a operação completou e prepara o usuário para o próximo passo.

Use um temporizador curto de dois a três segundos mostrando esta mensagem antes de navegar automaticamente para a tela de login. Alternativamente, você pode logar o usuário automaticamente com a nova senha e levá-lo direto para dentro do aplicativo, o que cria uma experiência ainda mais suave.

Tela Inicial: Auth Wrapper e Splash Screen

A tela inicial tem uma responsabilidade crítica: decidir rapidamente para onde levar o usuário baseado no estado de autenticação. Usuários retornando que já estão autenticados devem ir direto para a home do aplicativo. Usuários não autenticados devem ver a tela de login. Esta decisão precisa ser rápida e silenciosa, sem confundir o usuário com telas piscando.

Auth Wrapper: O Diretor de Tráfego

O Auth Wrapper é um widget que escuta o estado de autenticação e renderiza a tela apropriada. Ele não tem interface própria, apenas decide qual outra tela mostrar:

```dart
class AuthWrapper extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final authState = ref.watch(authProvider);

    // Durante verificação inicial, mostra splash
    if (authState is AuthStateInitial) {
      return SplashScreen();
    }

    // Se autenticado, vai para home
    if (authState is AuthStateAuthenticated) {
      return HomePage();
    }

    // Se não autenticado, mostra login
    return LoginPage();
  }
}
```

Este padrão é elegante porque centraliza a lógica de navegação baseada em autenticação. Quando o estado muda de não autenticado para autenticado após login, o wrapper automaticamente reconstrói e mostra HomePage. Quando o usuário faz logout e o estado volta para não autenticado, automaticamente mostra LoginPage novamente. Não há código de navegação espalhado por múltiplas telas.

Splash Screen: Enquanto Verifica

Durante o estado inicial, enquanto o aplicativo verifica se há sessão válida salva, mostre uma splash screen simples. Esta tela deve ser minimalista: fundo com cor ou gradiente da marca, logo centralizado, e talvez um pequeno indicador de progresso. Não coloque muita informação aqui porque esta tela deve aparecer apenas por uma fração de segundo.

A splash screen serve dois propósitos. Primeiro, evita mostrar brevemente a tela de login para usuários que estão autenticados, o que seria visualmente confuso. Segundo, dá ao aplicativo tempo para carregar recursos iniciais e verificar autenticação sem tela em branco.

Mantenha esta verificação rápida. Se demora mais que um ou dois segundos, há algo errado com sua implementação. A verificação de sessão do Supabase, por exemplo, é praticamente instantânea porque apenas lê do armazenamento local e valida o token.

Transições e Navegação Entre Telas

Como o usuário move entre estas telas é tão importante quanto o design das telas em si. Transições bruscas ou inesperadas quebram a ilusão de um aplicativo coeso e podem confundir ou frustrar usuários.

Navegação Push vs. Replacement

Quando o usuário clica em "Cadastre-se" na tela de login, use navegação normal com push. Isto coloca a tela de cadastro sobre a de login, mantendo a de login na pilha de navegação. O usuário pode pressionar voltar para retornar ao login, o que faz sentido conceitualmente.

Porém, quando o usuário completa login com sucesso, use pushReplacement ao invés de push. Isto substitui a tela de login pela home, removendo o login da pilha. O usuário não pode voltar para a tela de login pressionando o botão voltar, o que é correto porque ele está autenticado agora. Se ele pressionar voltar da home, o app fecha ao invés de voltar para login.

```dart
// Navegação normal para cadastro (pode voltar)
Navigator.of(context).push(
  MaterialPageRoute(builder: (_) => RegisterPage()),
);

// Navegação de substituição após login (não pode voltar)
Navigator.of(context).pushReplacement(
  MaterialPageRoute(builder: (_) => HomePage()),
);
```

Animações de Transição

As transições padrão do Flutter são boas, mas você pode customizar para criar uma experiência mais polida. Para telas de autenticação, transições suaves e não distrativas funcionam melhor. Um fade simples ou slide sutil são apropriados. Evite animações muito elaboradas ou longas que fazem o usuário esperar.

Mensagens de Erro e Feedback

Como você comunica erros ao usuário nessas telas impacta significativamente a experiência. Erros acontecem e precisamos guiar o usuário na resolução sem frustração.

Onde Mostrar Erros

Para erros relacionados a um campo específico, mostre o erro diretamente abaixo do campo usando o sistema de validação do Form. O texto de erro aparece em vermelho abaixo do campo problemático, deixando claro exatamente o que precisa ser corrigido.

Para erros gerais não relacionados a um campo específico, como problemas de rede ou credenciais inválidas, use um SnackBar na parte inferior da tela. SnackBars são não-intrusivos mas chamativos o suficiente para serem notados, e desaparecem automaticamente após alguns segundos.

Linguagem das Mensagens

Mensagens de erro devem ser específicas, acionáveis e amigáveis. Ao invés de "Erro 401", diga "Email ou senha incorretos". Ao invés de "Network error", diga "Não foi possível conectar. Verifique sua internet e tente novamente". A mensagem deve dizer ao usuário exatamente o que deu errado e, quando possível, como resolver.

Evite jargão técnico ou códigos de erro que não significam nada para usuários leigos. Evite também linguagem que culpa o usuário como "Você digitou errado". Seja neutro ou até assuma a responsabilidade: "Não conseguimos completar o login" ao invés de "Você não conseguiu fazer login".

Conclusão da Seção

Nesta seção, exploramos em profundidade cada tela que compõe o fluxo de autenticação. Detalhamos não apenas os elementos visuais necessários, mas também o raciocínio por trás de cada decisão de design e implementação. Aprendemos como criar telas que são funcionais, intuitivas e transmitem profissionalismo.

Cada tela que desenhamos serve um propósito claro na jornada do usuário, e a forma como elas se conectam cria uma experiência coesa. O login é a porta de entrada, o cadastro é onde conquistamos novos usuários, a recuperação de senha é nossa rede de segurança para usuários esquecidos, e o auth wrapper orquestra tudo silenciosamente nos bastidores.

Na próxima seção, vamos focar em como tornar essas telas não apenas funcionais, mas verdadeiramente agradáveis de usar, explorando boas práticas de experiência do usuário e acessibilidade que elevam a qualidade da nossa implementação.

Fim da Seção 3: As Telas do Fluxo de Autenticação