Documento de Arquitetura: Fluxo de Logout Reativo

Este documento complementa o fluxo de login, explicando como o processo de logout é gerenciado de forma reativa, utilizando `provider` e `go_router`.

1. O Princípio do Logout Reativo

Assim como no login, o processo de logout não envolve uma navegação manual (ou seja, não há um comando direto como `context.go('/login')`). Em vez disso, a interface do usuário (a tela de menu) simplesmente **informa ao estado global que a sessão terminou**. A navegação é uma **consequência automática** dessa mudança de estado, gerenciada inteiramente pelo `GoRouter`.

O fluxo é mais simples que o de login, pois envolve menos etapas.

2. O Fluxo de Logout: A Cadeia de Eventos Inversa

Aqui está o passo a passo do que acontece quando o usuário clica no botão "Sair".

Passo 1: A View (Tela de Menu)

- **Arquivo**: `lib/presentation/features/menu/view/menu_screen.dart`
- **Ação**: O usuário clica no `IconButton` de "Sair" na `AppBar`. Para confirmar a intenção, um diálogo de confirmação é exibido. Se o usuário confirmar, a ação de logout é disparada.
- **Código Chave**:

```
// Dentro do onPressed do botão "Sair" no AlertDialog
onPressed: () {
  // Ação principal: Encontra o SessionViewModel e chama o método logout.
  context.read<SessionViewModel>().logout();

  // Fecha o diálogo de confirmação.
  Navigator.of(dialogContext).pop();
},
```

- **Responsabilidade**: Capturar a intenção de logout do usuário. A View não sabe o que acontece depois; ela apenas notifica o `SessionViewModel` que a sessão deve ser encerrada. Ela usa `context.read` porque está apenas *executando uma ação*, e não precisa reconstruir a tela se o `SessionViewModel` mudar.

Passo 2: O ViewModel de Sessão (Estado Global)

- **Arquivo**: `lib/core/viewmodels/session_viewmodel.dart`
- **Ação**: O método `logout` é chamado pela `MenuScreen`. Ele limpa os dados do usuário e, crucialmente, notifica os "ouvintes" sobre a mudança.
- **Código Chave**:

```dart
void logout() {
  if (_currentUser != null) {
    _currentUser = null;

    // Grita para todos os "ouvintes" que o estado da sessão mudou.
    notifyListeners();
  }
}
```

- **Responsabilidade**: Atualizar a fonte da verdade do estado de autenticação. Ao definir `_currentUser` como `null`, a propriedade `isLoggedIn` automaticamente se torna `false`. A chamada a `notifyListeners()` é o gatilho para o próximo passo.

Passo 3: O Roteador (O Cérebro da Navegação)

- **Arquivo**: `lib/navigation/app_router.dart`
- **Ação**: O `GoRouter`, que está ouvindo o `SessionViewModel` (`refreshListenable`), "acorda" com a chamada do `notifyListeners()` e reavalia sua lógica de `redirect`.
- **Código Chave**:

```dart
late final router = GoRouter(
  // 1. O GoRouter ouve a notificação.
  refreshListenable: _sessionViewModel,

  // 2. A lógica de redirect é re-executada.
  redirect: (context, state) {
    final bool isLoggedIn = _sessionViewModel.isLoggedIn; // Agora é `false`
    final String currentLocation = state.matchedLocation;   // É '/menu'

    // 3. A SEGUINTE CONDIÇÃO AGORA SE TORNA VERDADEIRA:
    if (!isLoggedIn && currentLocation != '/login') {
      // "Se o usuário NÃO está logado E está em uma página que NÃO é a de login..."

      // 4. A regra retorna a string do caminho de destino.
      return '/login';
    }
    return null;
  },
);
```

- **Responsabilidade**: Agir como o guarda de trânsito. Ele detecta que o usuário não está mais logado (`isLoggedIn` é `false`), mas ainda está em uma rota protegida (`/menu`). A regra de `redirect` é clara: nesse caso, o usuário deve ser redirecionado para a tela de login. O `GoRouter` então executa a navegação, removendo a tela de menu da pilha e exibindo a tela de login.

3. Conclusão do Ciclo

O fluxo de logout demonstra a força e a elegância da arquitetura reativa:

- **Simplicidade na UI**: A tela de menu não precisa se preocupar com a lógica de navegação. Ela apenas dispara um evento.
- **Consistência**: A mesma lógica de `redirect` que protege as rotas contra acesso indevido é a que lida com o logout, garantindo que as regras de acesso sejam aplicadas de forma consistente em todo o aplicativo.
- **Clareza**: O fluxo de dados é unidirecional e fácil de seguir: **Ação da UI -> Atualização do Estado Global -> Reação da Navegação**.

Entender o ciclo completo de login e logout solidifica a compreensão do padrão `Provider` + `GoRouter` como uma base poderosa para qualquer aplicativo Flutter.