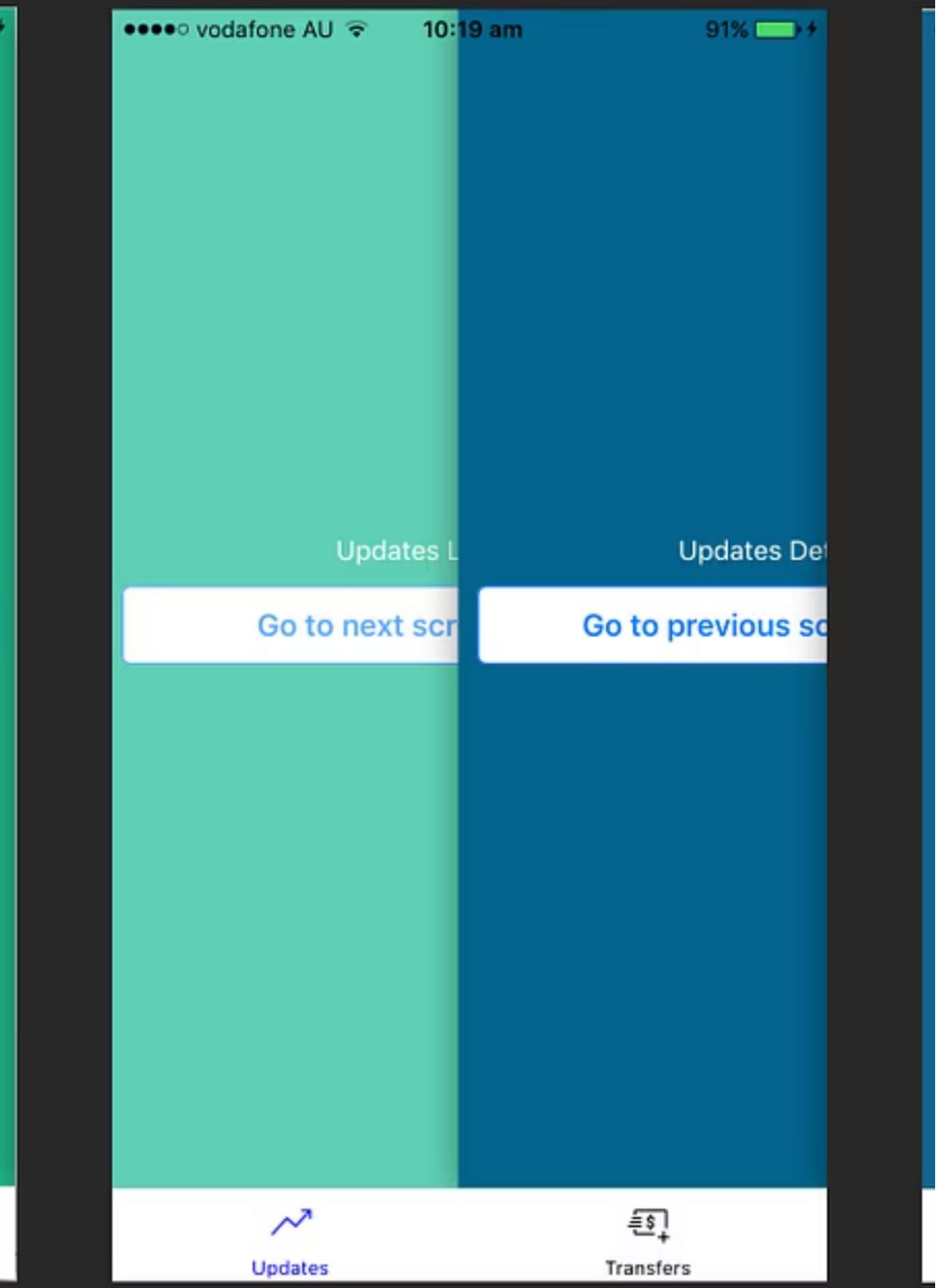
# Desenvolvimento de Aplicativo Mobile

Material base desenvolvido pelo
Prof. Raphael Barreto

raphael.b.oliveira@docente.senai.br

2026
SENAI - SAPUCAÍ




# Navegação em Stack e Empilhamento de Telas

Desenvolvimento Mobile com React Native e Expo Router

SENAI 2026

MÓDULO 9

# Sumário

CAPÍTULO 01

# Introdução à Navegação Mobile

Compreendendo os fundamentos da navegação em aplicativos móveis e suas diferenças em relação ao desenvolvimento web tradicional

# Navegação Web vs Mobile: Diferenças Fundamentais

A navegação em aplicativos móveis apresenta características distintas da navegação em navegadores web. Enquanto na web utilizamos URLs e o histórico do navegador, nos dispositivos móveis trabalhamos com conceitos como **pilhas de telas** (stacks), **abas** (tabs) e **gavetas** (drawers).

No desenvolvimento web com React, estamos acostumados a usar bibliotecas como React Router DOM para gerenciar rotas. No React Native, utilizamos o **Expo Router**, que oferece uma abordagem baseada em arquivos, onde a estrutura de pastas define automaticamente as rotas da aplicação.

# Tipos de Navegação em Aplicativos Móveis

### Stack Navigation

Sistema de empilhamento de telas, permitindo navegação sequencial com histórico

### Tab Navigation

Navegação por abas na parte inferior ou superior da tela

### Drawer Navigation

Menu lateral deslizante com opções de navegação

Nesta aula, vamos focar no tipo de navegação mais próximo do que conhecemos no desenvolvimento web: a **Stack Navigation**. Este é o padrão mais comum em aplicativos móveis e serve como base para entender os demais tipos.

A stack funciona como uma pilha de pratos: você adiciona telas no topo e pode removê-las voltando para as anteriores. Esse comportamento natural está presente na maioria dos aplicativos que você usa diariamente.

# O Expo Router: Roteamento Baseado em Arquivos

O Expo Router revoluciona a forma como criamos navegação em aplicativos React Native ao adotar um sistema de roteamento baseado em arquivos, similar ao que você encontra em frameworks como Next.js.

## Vantagens da Abordagem

- Estrutura intuitiva e fácil de entender
- Menos código de configuração necessário
- Rotas definidas automaticamente pela estrutura de pastas
- Deep linking nativo sem configuração adicional
- Melhor organização e manutenção do código

## Como Funciona

Cada arquivo na pasta `app/` se torna automaticamente uma rota. O nome do arquivo define o caminho da URL, e a estrutura de pastas cria rotas aninhadas naturalmente.

O arquivo especial `_layout.jsx` permite configurar a navegação e compartilhar elementos visuais entre diferentes telas.

CAPÍTULO 02

# Configuração Inicial do Projeto

Preparando a estrutura de arquivos e organizando os componentes para implementar a navegação

# Estrutura de Pastas no Expo Router

A organização de arquivos em um projeto Expo Router segue convenções específicas que diferem do React tradicional. Compreender essa estrutura é fundamental para o desenvolvimento eficiente.

A pasta `app/` é o coração da aplicação, onde residem todas as telas e configurações de navegação. Diferentemente de colocar páginas em uma pasta `components/` ou `pages/`, aqui elas ficam diretamente na raiz do app, e seus nomes de arquivo definem as rotas.

# Convenções de Nomenclatura de Arquivos

### index.jsx

A página inicial da aplicação, equivalente à rota `/`. Este é sempre o primeiro arquivo carregado quando o app abre.

### _layout.jsx

Arquivo especial que configura a navegação e elementos compartilhados. O underscore indica que não é uma rota acessível.

### nome-da-tela.jsx

Arquivos de telas individuais. Use **letras minúsculas** e separe palavras com hífen. O nome do arquivo é exatamente o nome da rota.

**Importante:** Diferente dos componentes React tradicionais que usam PascalCase (ex: HomePage.jsx), as telas no Expo Router devem usar kebab-case (ex: home-page.jsx) porque o nome do arquivo se torna parte da URL.

# Criando os Arquivos Iniciais

Para começar, vamos criar a estrutura básica do nosso projeto com os seguintes arquivos na pasta **app/**:

- `_layout.jsx` - Configuração da navegação
- `index.jsx` - Tela inicial (Home)

**App/components/**:

- `product-detail.jsx` - Detalhes do produto
- `products.jsx` - Tela de produtos
- `settings.jsx` - Tela de configurações
- `user.jsx` - Tela de usuário

**styles/**:

- `styles.js` - estilo do projeto

Cada arquivo conterá um componente simples com texto e cor de fundo diferente para facilitar a identificação durante os testes de navegação.

# Criando o componente _layout

Para nossos testes iniciais, vamos criar componentes básicos que exibem apenas texto e uma cor de fundo. Isso nos permite focar na navegação sem nos preocuparmos com conteúdo complexo.

## Exemplo: _layout.jsx (configuração de navegação)

```
import { Text } from "react-native"

export default function RootLayout(){
  return <Text>Navegação</Text>
}
```

# Criando Componentes de Tela Simples

Para nossos testes iniciais, vamos criar componentes básicos que exibem apenas texto e uma cor de fundo. Isso nos permite focar na navegação sem nos preocuparmos com conteúdo complexo.

## Exemplo: index.jsx (Home)

```
import { Text, View } from "react-native"
import { styles } from "../styles/styles"

export default function Home() {
  return (
    <View style={[styles.container,
      { backgroundColor: "#FAEDCB" }]}>
      <Text>Home</Text>
    </View>
  )
}
```

Cada tela usa o mesmo padrão: um `View` centralizado com uma cor de fundo única e um `Text` identificando a tela.

As cores diferentes (amarelo para Home, azul para Settings, verde para User) facilitam a identificação visual ao navegar entre as telas durante o desenvolvimento.

# Arquivo de Estilos Compartilhados

Vamos criar um arquivo `styles/styles.js` com estilos reutilizáveis para todas as nossas telas, mantendo a consistência visual e facilitando a manutenção.

```
import { StyleSheet } from "react-native"

export const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center"
  }
})
```

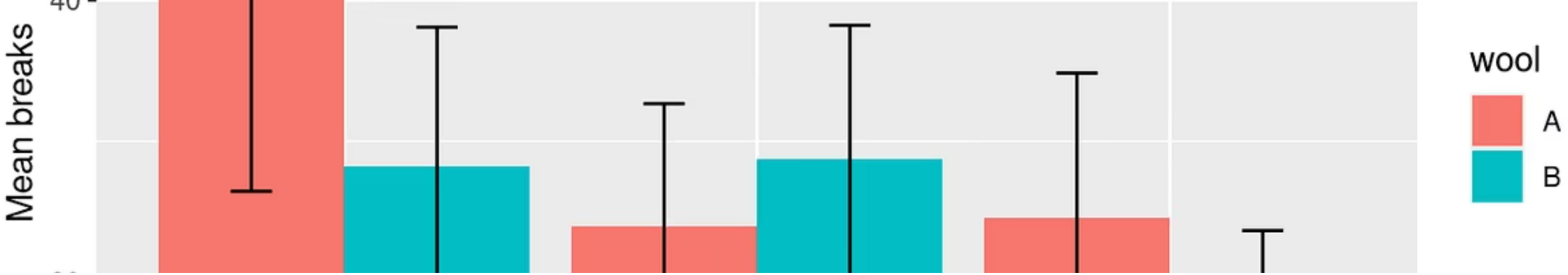
O estilo `container` centraliza o conteúdo tanto horizontal quanto verticalmente usando Flexbox. Este é um padrão muito comum em aplicativos móveis.

A propriedade `flex: 1` faz com que o componente ocupe todo o espaço disponível na tela, essencial para que o layout funcione corretamente.

CAPÍTULO 03

# Implementando Stack Navigation

Configurando o sistema de navegação em pilha e conectando todas as telas do aplicativo




# O Conceito de Stack (Pilha)

A navegação em stack funciona exatamente como uma pilha de pratos: você adiciona itens no topo (push) e remove sempre do topo também (pop). Esse é o comportamento padrão da navegação "para frente" e "para trás" nos aplicativos móveis.

## Características da Stack

- Mantém histórico de navegação
- Botão "voltar" automático
- Transições animadas entre telas
- Gestão eficiente de memória
- Estado preservado ao navegar

## Como Funciona

Quando você navega de uma tela para outra, a nova tela é "empilhada" sobre a anterior. A tela anterior permanece na memória, mas não visível. Ao pressionar voltar, a tela do topo é removida, revelando a anterior.

# Configurando o _layout.jsx

O arquivo `_layout.jsx` é onde configuramos toda a estrutura de navegação. Este arquivo é executado primeiro e envolve todas as outras telas da aplicação.

## Código Inicial Básico

```
import { Stack } from "expo-router"

export default function RootLayout() {
  return (
    <Stack>

    </Stack>
  )
}
```

Começamos importando o componente `Stack` do Expo Router. Este componente é responsável por gerenciar toda a navegação em pilha.

O nome da função (`RootLayout`) pode ser qualquer um, mas o nome do arquivo deve ser exatamente `_layout.jsx` para que o Expo Router o reconheça.

# Registrando Telas na Stack

Para cada tela que queremos disponibilizar na navegação, precisamos adicionar um `Stack.Screen` dentro do componente `Stack`. O atributo `name` deve corresponder exatamente ao nome do arquivo na pasta `app/`.

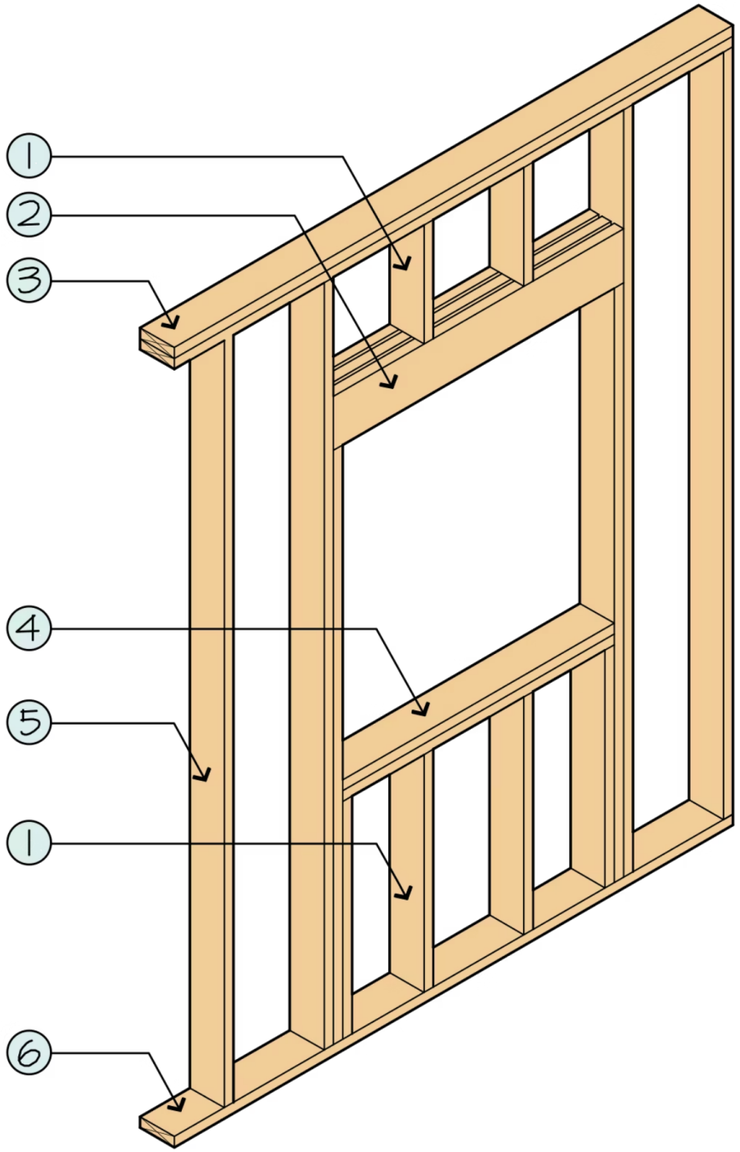
```
import { Stack } from "expo-router"

export default function RootLayout() {
  return (
    <Stack>
      <Stack.Screen name="index" />
      <Stack.Screen name="componentes/settings" />
      <Stack.Screen name="componentes/user" />
    </Stack>
  )
}
```

## Pontos Importantes

- O `name` deve ser idêntico ao nome do arquivo (sem .jsx)
- A ordem dos `Stack.Screen` não afeta a navegação
- Apenas telas registradas aqui podem ser navegadas
- O arquivo `index` representa a rota raiz `/`

Se você acabou de criar as rotas e não está vendo mudanças, pode ser necessário recarregar o simulador ou reiniciar o servidor de desenvolvimento.




# O Header Padrão

Ao configurar a Stack, você perceberá que automaticamente aparece uma barra no topo de cada tela. Esta é a **Navigation Bar** (ou Header), um elemento padrão dos aplicativos móveis que fornece contexto sobre onde o usuário está e permite navegação.

## Elementos do Header

- **Título:** Nome da tela atual
- **Botão Voltar:** Aparece automaticamente quando há histórico
- **Ações:** Espaço para botões adicionais

Por padrão, o título mostrado é o nome do arquivo, o que nem sempre é ideal. Por exemplo, o arquivo `index` mostraria "index" como título, mas seria mais apropriado mostrar "Home".

Felizmente, podemos personalizar facilmente o título e outros aspectos visuais do header.

# Personalizando Títulos das Telas

Para alterar o título exibido no header, utilizamos o atributo `options` em cada `Stack.Screen`. Isso nos permite definir um título mais amigável do que o nome do arquivo.

```
<Stack>
  <Stack.Screen name="index" options={{headerTitle: "Home"}}/>
  <Stack.Screen name=“componentes/settings" options={{headerTitle: "Configurações"}}/>
  <Stack.Screen name=componentes/user" options={{headerTitle: "Usuário"}}/>
</Stack>
```

O atributo `options` aceita um objeto com diversas configurações. `headerTitle` define o texto que aparece no centro do header.

Agora, em vez de ver "index", "settings" e "user", o usuário verá títulos mais descritivos em português: "Home", "Configurações" e "Usuário".

Esta é apenas uma das muitas opções de personalização disponíveis para o header.

CAPÍTULO 04

# Personalização Visual da Navegação

Customizando cores, estilos e aparência dos elementos de navegação para criar uma identidade visual consistente


# Entendendo as Barras do Aplicativo

Um aplicativo móvel possui duas barras importantes na parte superior da tela, cada uma com função e configuração distintas. É fundamental entender a diferença entre elas para personalizar corretamente.

## Status Bar

A barra no topo absoluto da tela que mostra informações do sistema: hora, bateria, sinal Wi-Fi, notificações. Esta barra é controlada pelo sistema operacional.

- Sempre visível
- Informações do sistema
- Customizável via código

## Navigation Bar (Header)

A barra logo abaixo da Status Bar, específica do aplicativo. Mostra o título da tela atual, botão voltar e ações disponíveis. Totalmente configurável pelo desenvolvedor.

- Parte do aplicativo
- Título e navegação
- Totalmente customizável

# Configurando Estilos Globais do Header

Em vez de configurar cada tela individualmente, podemos definir estilos globais usando `screenOptions` no componente `Stack`. Essas configurações se aplicam a todas as telas da stack.

```
<Stack screenOptions={{headerStyle: {backgroundColor: "#E94560"}, headerTintColor: "#FFFFFF"}}>
  <Stack.Screen name="index" options={{ headerTitle: "Home" }} />
  <Stack.Screen name="componentes/settings" options={{ headerTitle: "Configurações" }} />
  <Stack.Screen name="componentes/user" options={{ headerTitle: "Usuário" }} />
</Stack>
```

## Propriedades Importantes

`headerStyle` - Define estilos do container do header, como cor de fundo, altura, sombras, etc.

`headerTintColor` - Define a cor do texto e ícones no header (título, botão voltar).

Note que `headerTintColor` fica no mesmo nível que `headerStyle`, não dentro dele.

# Ocultando o Header

Em alguns casos, você pode querer ocultar completamente o header, especialmente para telas de splash, login ou telas com navegação customizada.

## Ocultar Globalmente

```
<Stack
  screenOptions={{
    headerShown: false
  }}
>
```

Esta configuração oculta o header de todas as telas.

> **Atenção:** Ocultar o header remove também o botão de voltar. Certifique-se de fornecer uma forma alternativa de navegação quando necessário.

# Personalizando a Status Bar

A Status Bar é controlada através do componente `StatusBar` do Expo. Podemos personalizar sua aparência para harmonizar com o design do aplicativo.

```
import { Stack } from "expo-router"
import { StatusBar } from "expo-status-bar"

export default function RootLayout() {
  return (
    <>
      <StatusBar
        style="light"
        backgroundColor="#E94560"
      />
      <Stack
        screenOptions={{ headerStyle: { backgroundColor: "#E94560" }, headerTintColor: "#FFFFFF", headerShown: true }}>
        <Stack.Screen name="index" options={{ headerTitle: "Home" }} />
        <Stack.Screen name="componentes/settings" options={{ headerTitle: "Configurações" }} />
        <Stack.Screen name="componentes/user" options={{ headerTitle: "Usuário" }} />
      </Stack>
    </>
  )
}
```

## Propriedades da Status Bar

`style` - Define se os ícones serão claros ("light") ou escuros ("dark"). Use "light" para fundos escuros e "dark" para fundos claros.

`backgroundColor` - Cor de fundo da Status Bar. Funciona apenas no Android; no iOS, a Status Bar é sempre translúcida.

Usamos um Fragment (`<>`) para envolver ambos os componentes, já que o `return` só pode retornar um elemento raiz.

# Harmonizando as Cores

Para criar uma experiência visual coesa, é importante que a Status Bar e o Header compartilhem a mesma cor ou cores complementares. Isso cria continuidade visual e profissionalismo.

### Status Bar

Cor de fundo + ícones claros/escuros

```
backgroundColor="#E94560"style="light"
```

### Header

Mesma cor de fundo + texto branco

```
backgroundColor="#E94560"headerTintColor="#FFFFFF"
```

### Resultado

Interface unificada e profissional

As duas barras parecem uma única barra integrada

CAPÍTULO 05

# Navegando Entre Telas

Implementando navegação com Links e Router: duas abordagens para mover-se entre as telas do aplicativo

# Duas Formas de Navegar

O Expo Router oferece duas abordagens principais para navegar entre telas, cada uma adequada para situações diferentes. Entender quando usar cada uma é fundamental para criar uma experiência de usuário fluida.

## Componente Link

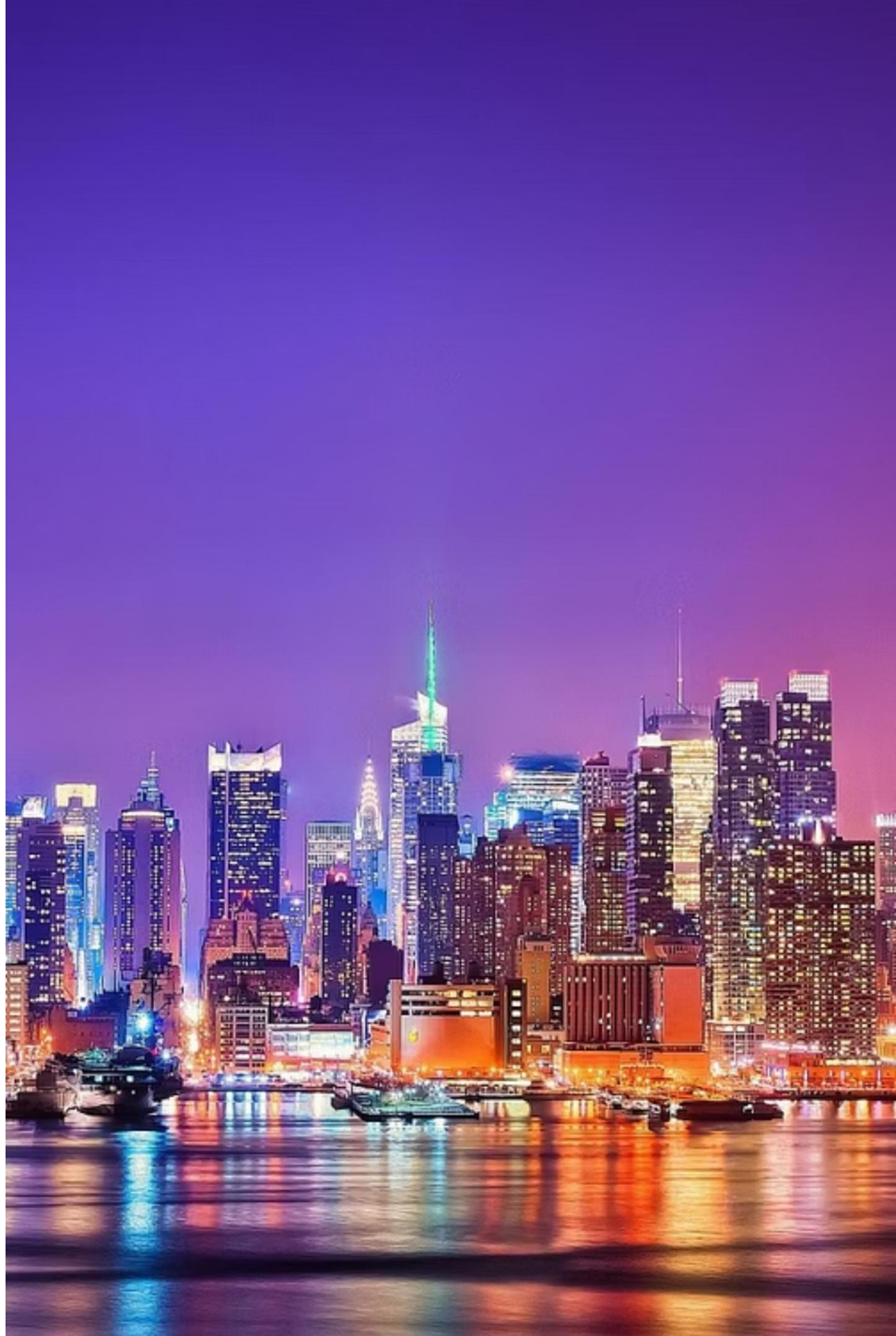
Usado quando a navegação é iniciada diretamente pelo usuário através de um toque ou clique. Similar à tag `<a>` do HTML.

- Navegação declarativa
- Acionada por interação do usuário
- Mais simples de implementar
- Ideal para botões e menus

## Router Programático

Usado quando a navegação precisa acontecer após alguma lógica, como validação, requisição de API ou processamento de dados.

- Navegação imperativa
- Acionada por código
- Mais flexível e poderosa
- Ideal para fluxos complexos



# Navegação com Link

O componente `Link` é a forma mais simples e direta de criar navegação. Funciona de maneira similar aos links de páginas web, mas otimizado para aplicativos móveis. Vamos inserir o código na página index.jsx:

## Implementação Básica

```
import { Text, View } from "react-native"
import { styles } from "../styles/styles"
import { Link } from "expo-router"

export default function Home() {
  return (
    <View style={[styles.container, { backgroundColor: "#FAEDC8" }]}>
      <Text>Home</Text>
      <Link href="/components/user">Ir para Usuários</Link>
    </View>
  )
}
```

# Estrutura de Rotas

É importante compreender como os nomes de arquivos se traduzem em rotas para usar o Link corretamente.

1. **index.jsx**

   Rota: `/`

2. **settings.jsx**

   Rota: `/settings`

3. **user.jsx**

   Rota: `/user`

4. **product-detail.jsx**

   Rota: `/product-detail`

O arquivo `index.jsx` é especial: ele representa a rota raiz `/`. Todos os outros arquivos criam rotas com seus próprios nomes.

Nomes com hífen se tornam rotas com hífen. O nome do arquivo deve corresponder exatamente ao que você coloca no `href`, incluindo maiúsculas e minúsculas (embora a convenção seja usar minúsculas).

# Encadeando Navegação

Você pode criar cadeias de navegação colocando Links em várias telas diferentes. Cada navegação adiciona uma nova tela à pilha, permitindo que o usuário volte através do histórico.

## De Home para User

```
// index.jsx
<Link href=“/components/user">Ir para Usuários</Link>
```

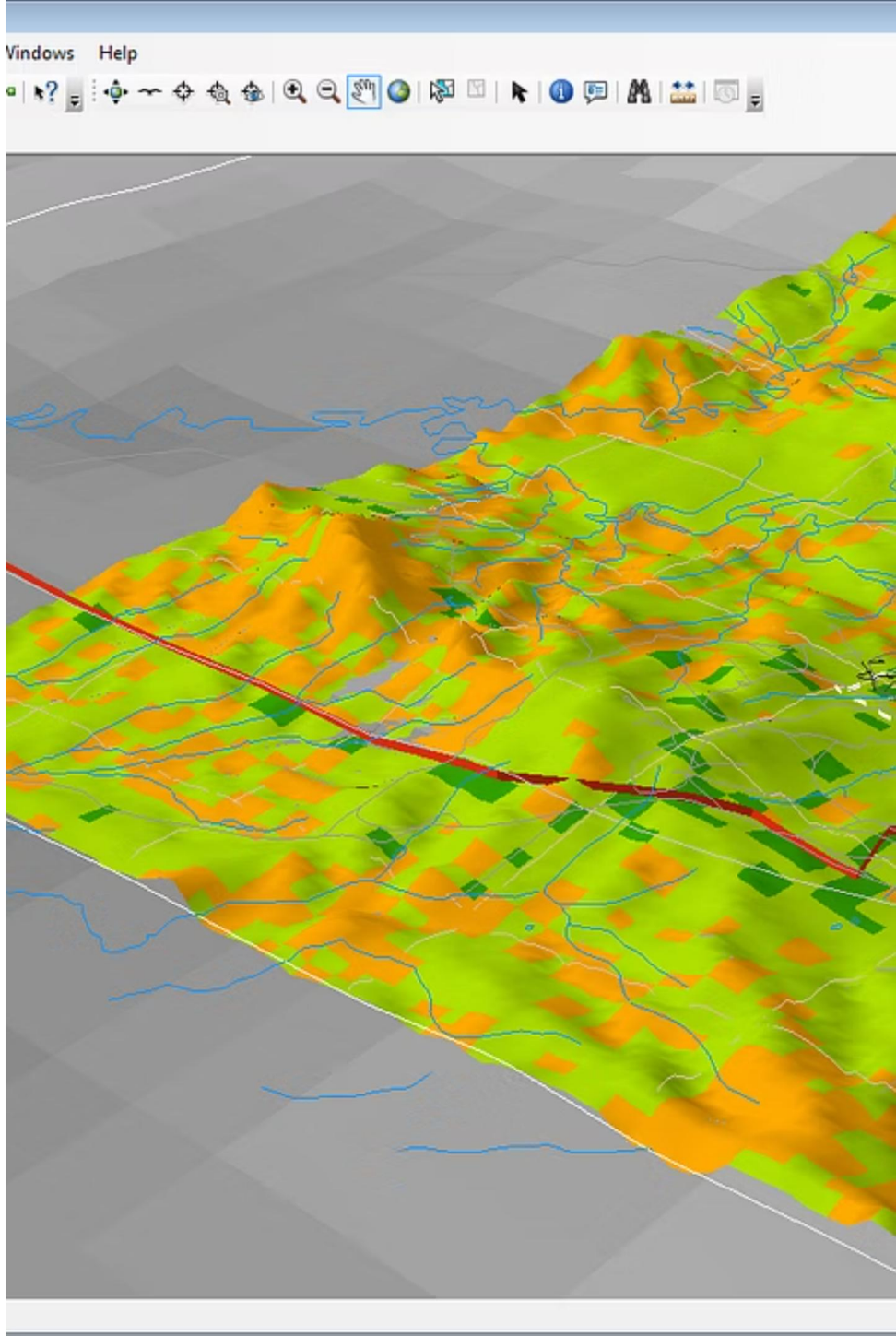
## De User para Settings

```
// user.jsx
<Link href=“/components/settings">Ir para Configurações</Link>
```

# Navegação Programática com Router

Quando você precisa navegar após executar alguma lógica - como validar um formulário, fazer uma requisição, ou processar dados - o router programático é a solução ideal.

O objeto `router` fornece métodos para controlar a navegação através de código JavaScript, oferecendo mais flexibilidade do que o componente Link.



# Implementando Router Programático (settings)

```
import { Pressable, Text, View } from "react-native"
import { styles } from "../../styles/styles"
import { router } from "expo-router"

export default function Settings() {
    const goToHome = () => {
        router.navigate("/")
    }
    return (
        <View style={[styles.container, { backgroundColor: "#FAC124FF" }]}>
            <Text>Configurações</Text>
            <Pressable onPress={goToHome}>
                <Text>Voltar para Home</Text>
            </Pressable>
        </View>
    )
}
```

# Entendendo a Navegação Programática

O exemplo anterior demonstra como o objeto `router` nos permite controlar a navegação do aplicativo de forma imperativa, acionando transições de tela após a execução de alguma lógica ou evento.

### Importação do Router

A linha `import { router } from "expo-router"` é fundamental para ter acesso aos métodos de navegação programática.

### Função de Navegação

A função `goToHome` encapsula a lógica de navegação, sendo disparada pelo componente `Pressable`.

### Método `navigate`

`router.navigate("/")` é o comando que direciona o usuário para a rota especificada (neste caso, a rota raiz que corresponde à tela "Home").

### Interação com `Pressable`

O componente `Pressable`, junto com o evento `onPress`, permite que o usuário acione a navegação programática ao interagir com o elemento.

CAPÍTULO 06

# Métodos de Navegação

Navigate, Push e Replace: entendendo as diferenças e quando usar cada método

# Três Formas de Adicionar Telas à Pilha

O Expo Router oferece três métodos principais para navegar entre telas, cada um com comportamento diferente em relação à pilha de navegação. A escolha do método correto impacta diretamente a experiência do usuário.

# Navigate: Reutilizando Telas na Pilha

O método `navigate` é inteligente: ele verifica se a tela de destino já existe na pilha. Se existir, remove todas as telas acima dela e retorna para aquela instância. Se não existir, adiciona uma nova.

## Como Funciona

```
router.navigate("/settings")
```

Estado da pilha: `[Home, User, Settings]`
Executando: `router.navigate("/user")`

Resultado: `[Home, User]`

A tela Settings foi removida e voltamos para o User que já existia.

## Quando Usar

- Navegação em menus principais
- Evitar duplicação de telas
- Comportamento similar à navegação web
- Quando você quer "limpar" o histórico

## Vantagens

- Previne pilhas muito profundas
- Economiza memória
- Comportamento previsível

# Push: Sempre Adicionando Nova Tela

O método `push` sempre adiciona uma nova instância da tela no topo da pilha, mesmo que ela já exista em outro lugar. Cada push cria uma nova entrada no histórico.

## Como Funciona

```
router.push("/settings")
```

Estado da pilha: `[Home, User]`
Executando: `router.push("/user")`

Resultado: `[Home, User, User]`

Uma nova instância de User foi adicionada, mesmo já existindo uma na pilha.

## Quando Usar

- Navegação linear (passo a passo)
- Formulários multi-etapas
- Quando cada visita é independente
- Padrão para aplicativos móveis

## Características

- Botão voltar sempre funciona
- Histórico completo preservado
- Pode criar pilhas profundas

**Recomendação:** Quando em dúvida, use `push`. É o comportamento mais comum e esperado em aplicativos móveis, garantindo que o usuário sempre possa voltar.

# Replace: Substituindo a Tela Atual

O método `replace` substitui a tela atual pela nova, removendo a atual da pilha. Isso impede que o usuário volte para a tela anterior usando o botão voltar.

## Como Funciona

```
router.replace("/home")
```

Estado da pilha: `[Home, Login, Loading]`
Executando replace em Loading: `router.replace("/dashboard")`

Resultado: `[Home, Login, Dashboard]`

Loading foi removida e Dashboard entrou em seu lugar.

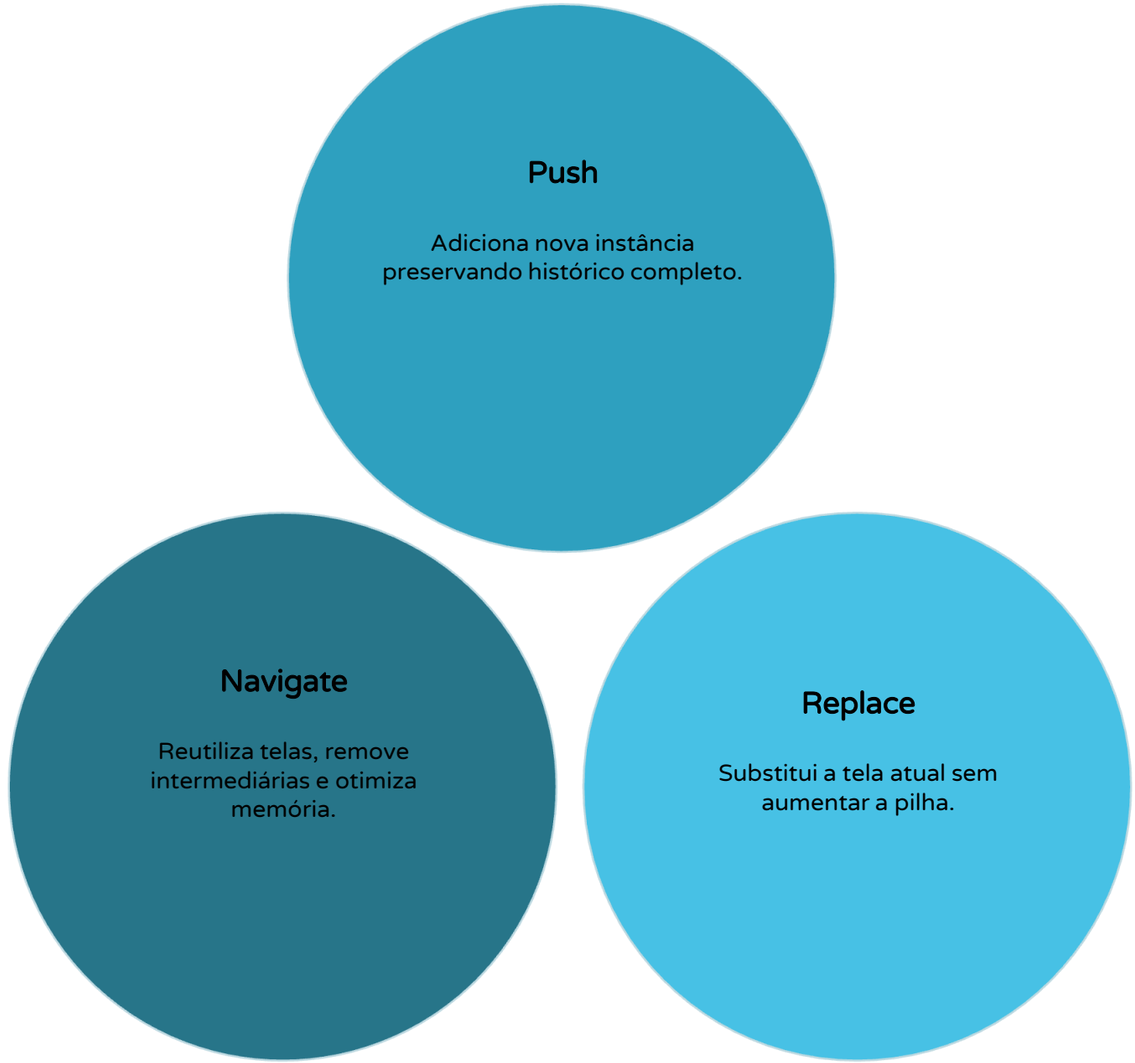
## Quando Usar

- **Após login/logout:** não voltar para tela de login
- **Após splash screen:** não voltar para a tela inicial
- **Após erro:** não voltar para tela de erro
- **Fluxos irreversíveis:** confirmação de pagamento

## Importante

Use com cautela! Impede navegação reversa pode confundir usuários se não for intencional.

# Comparação Visual dos Métodos



# Aplicando Métodos no Link

Os três métodos de navegação também estão disponíveis no componente Link, não apenas no router programático. Basta adicionar o atributo correspondente.

## Navigate (padrão)

```
<Link href="/user">
  Ir para Usuários
</Link>
```

Sem especificar método, o Link usa navigate por padrão.

## Push

```
<Link href="/user" push>
  Ir para Usuários
</Link>
```

Adiciona nova instância da tela.

## Replace

```
<Link href="/home" replace>
  Voltar ao início
</Link>
```

Substitui a tela atual.

A escolha entre usar Link ou router programático depende da sua necessidade: Link para navegação direta, router para navegação com lógica adicional.

CAPÍTULO 07

# Código Completo do Projeto

Implementação prática passo a passo de todos os conceitos apresentados

# Estrutura Final do Projeto

Agora que compreendemos todos os conceitos, vamos ver o código completo e funcional do projeto. Esta implementação integra navegação em stack, personalização visual e diferentes métodos de navegação.

**Pasta styles**

styles.js com temas e estilos

**Pages**

index, settings, user, products, product-detail

**_layout.jsx**

Define layout e navegação global

**Pasta app**

Contém páginas e layouts principais

# Arquivo: app/_layout.jsx

Este é o arquivo principal que configura toda a navegação do aplicativo, incluindo a stack, personalização do header e da status bar.

```
import { Stack } from "expo-router"
import { StatusBar } from "expo-status-bar"

export default function RootLayout() {
  return (
    <>
      <StatusBar style="light" backgroundColor="#E94560" />
      <Stack screenOptions={{ headerStyle: { backgroundColor: "#E94560" }, headerTintColor: "#FFFFFF" }}>
        <Stack.Screen name="index" options={{ headerTitle: "Home" }} />
        <Stack.Screen name="components/settings" options={{ headerTitle: "Configurações" }}/>
        <Stack.Screen name="components/user" options={{ headerTitle: "Usuário" }}/>
        <Stack.Screen name="components/products" options={{ headerTitle: "Produtos" }} />
        <Stack.Screen name="components/product-detail" options={{ headerTitle: "Detalhes do Produto" }} />
      </Stack>
    </>
  )
}
```

# Arquivo: styles/styles.js

Arquivo de estilos compartilhado entre todas as telas, promovendo consistência visual e facilitando manutenção.

```
import { StyleSheet } from "react-native"

export const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center"
  }
})
```

## Explicação dos Estilos

`flex: 1` - Faz o container ocupar todo o espaço disponível

`justifyContent: "center"` - Centraliza verticalmente

`alignItems: "center"` - Centraliza horizontalmente

Este é um padrão muito comum em apps mobile para centralizar conteúdo na tela.

# Arquivo: app/index.jsx (Home)

A tela inicial do aplicativo, servindo como ponto de entrada. Inclui navegação usando o componente Link com método push.

```jsx
import { Text, View } from "react-native"
import { styles } from "../styles/styles"
import { Link } from "expo-router"

export default function Home() {
  return (
    <View style={[styles.container, { backgroundColor: "#FAEDCB" }]}>
      <Text>Home</Text>
      <Link href="/components/user"> Ir para Usuários </Link>
      <Link href="/components/products"> Ir para Produtos </Link>
    </View>
  )
}
```

# Arquivo: app/user.jsx

Tela de usuário com navegação para a tela de configurações, também utilizando Link com push.

```
import { Text, View } from "react-native"
import { styles } from "../../styles/styles"
import { Link } from "expo-router"

export default function User() {
  return (
    <View style={[styles.container, { backgroundColor: "#F6B400FF" }]}>
      <Text>User</Text>
      <Link href="/components/settings"> Ir para Configurações </Link>
    </View>
  )
}
```

# Arquivo: app/settings.jsx

Tela de configurações demonstrando navegação programática com router. Inclui lógica customizada antes de navegar.

```
import { Pressable, Text, View } from "react-native"
import { styles } from "../../styles/styles"
import { router } from "expo-router"

export default function Settings() {
    const goToHome = () => {
        router.navigate("/")
    }
    return (
        <View style={[styles.container, { backgroundColor: "#FAC124FF" }]}>
            <Text>Configurações</Text>
            <Pressable onPress={goToHome}>
                <Text>Voltar para Home</Text>
            </Pressable>
        </View>
    )
}
```

# Diferença Entre Link e Router Neste Exemplo

## Usando Link (Home e User)

```
<Link href="/user" push>
  Ir para Usuários
</Link>
```

**Características:**

- Navegação imediata ao toque
- Mais simples e direto
- Ideal para navegação básica
- Menos código necessário

## Usando Router (Settings)

```
const goToHome = () => {
  // Lógica aqui
  router.push("/")
}

<Pressable onPress={goToHome}>
```

**Características:**

- Navegação após função executar
- Permite lógica adicional
- Ideal para fluxos complexos
- Mais controle sobre quando navegar

# Arquivos Adicionais: products.jsx

Tela de listagem de produtos. Embora não esteja conectada no fluxo principal, demonstra como adicionar novas telas ao projeto.

```
import { Text, View } from "react-native"
import { styles } from "../../styles/styles"
import { Link } from "expo-router"

export default function Products() {
  return (
    <View style={[styles.container, { backgroundColor: "#FAEDCB" }]}>
      <Text>Products</Text>
      <Link href="/components/product-detail"> Ver Detalhes do Produto </Link>
    </View>
  )
}
```

# Arquivos Adicionais: product-detail.jsx

Tela de detalhes do produto, demonstrando navegação hierárquica (lista → detalhe) comum em aplicativos.

```
import { Pressable, Text, View } from "react-native"
import { styles } from "../styles/styles"
import { router } from "expo-router"

export default function ProductDetail() {
    const goToHome = () => {
        router.navigate("/")
    }
        return (
            <View style={[styles.container, {
backgroundColor: "#FACE55FF" }]}>
                <Text>Detalhes do Produto</Text>
                <Pressable onPress={goToHome}>
                    <Text> Ir para Home</Text>
                </Pressable>
            </View>
        )
    }
```

## Padrão Lista-Detalhe

Este é um padrão muito comum em apps:

- Lista de itens (products)
- Clicar em um item
- Ver detalhes completos
- Voltar para a lista

Na próxima aula veremos como passar parâmetros para identificar qual produto está sendo visualizado.

# Recursos Adicionais e Documentação

## Documentação Oficial

- **Expo Router:** docs.expo.dev/router
- **React Navigation:** reactnavigation.org (base do Expo Router)
- **Expo SDK:** docs.expo.dev
- **React Native:** reactnative.dev

## Próximos Passos

- Rotas aninhadas e grupos de rotas
- Parâmetros de navegação (passar dados entre telas)
- Tab navigation e drawer navigation
- Deep linking e URLs personalizadas
- Animações de transição customizadas

# Conclusão e Próximos Passos

## O Que Você Aprendeu

- Fundamentos de navegação mobile
- Configuração completa do Expo Router
- Implementação prática de Stack Navigation
- Personalização visual avançada
- Diferentes métodos de navegação
- Boas práticas e otimização

## Continuando Seus Estudos

Esta aula cobriu os fundamentos essenciais da navegação em stack. Nas próximas aulas, você aprenderá conceitos mais avançados como:

- **Rotas Aninhadas:** Criar hierarquias complexas de navegação
- **Parâmetros de Rota:** Passar dados entre telas
- **Tab Navigation:** Navegação por abas na parte inferior
- **Drawer Navigation:** Menu lateral deslizante
- **Deep Linking:** Abrir telas específicas via URLs

Pratique os conceitos desta aula criando seu próprio aplicativo com múltiplas telas e diferentes fluxos de navegação. A prática é essencial para dominar o desenvolvimento mobile!

# Perguntas?

Entre em contato:

**raphael.b.oliveira@docente.senai.br**

Obrigado pela atenção! Estou à disposição para esclarecer dúvidas e ajudar no seu aprendizado.

Firjan SENAI