Desenvolvimento de Aplicativo Mobile

Material base desenvolvido pelo
Prof. Raphael Barreto

raphael.b.oliveira@docente.senai.br

2026

DEC - GEP

Gestos na Touch Screen e Animações

DESIGN DE INTERAÇÃO

REACT NATIVE

Curso de Desenvolvimento Mobile - SENAI 2026

Sumário

01

Tipos de Teclado Virtual

Configurações e Variações

O Teclado Virtual no Desenvolvimento Mobile

Diferentemente dos computadores que possuem um teclado físico único, os dispositivos móveis utilizam teclados virtuais programáveis. Essa característica oferece uma vantagem significativa: podemos personalizar o tipo de teclado que aparece para o usuário conforme o contexto de entrada de dados.

O React Native disponibiliza diversos tipos de teclado através da propriedade `keyboardType` no componente TextInput. Cada tipo é otimizado para um propósito específico, melhorando a experiência do usuário ao facilitar a entrada de informações.

Alguns tipos de teclado são exclusivos para iOS ou Android, mas existem opções compatíveis com ambas as plataformas, garantindo consistência na experiência do usuário.

Tipos de Teclado Multiplataforma

Default

Teclado padrão com letras, números e caracteres especiais. Aparece quando nenhum tipo específico é definido.

Number Pad

Teclado exclusivamente numérico, ideal para entrada de valores inteiros como códigos ou quantidades.

Decimal Pad

Inclui números e vírgula/ponto decimal, perfeito para valores monetários ou medidas precisas.

Email Address

Otimizado com acesso rápido ao @ e ponto, facilitando a digitação de endereços de e-mail.

Phone Pad

Teclado numérico com símbolos telefônicos como #, *, ideal para números de telefone.

Numeric

Variação numérica que pode incluir caracteres adicionais dependendo da plataforma.

KeyboardType

https://reactnative.dev/docs/textinput#keyboardtype

Implementando Tipos de Teclado

Sintaxe Básica

Para definir o tipo de teclado, adicione a propriedade `keyboardType` ao componente TextInput. O valor deve ser uma string correspondente ao tipo desejado.

A escolha adequada do tipo de teclado não é apenas uma questão de usabilidade — ela também previne erros de entrada ao limitar os caracteres disponíveis para o usuário.

Exemplo de Código

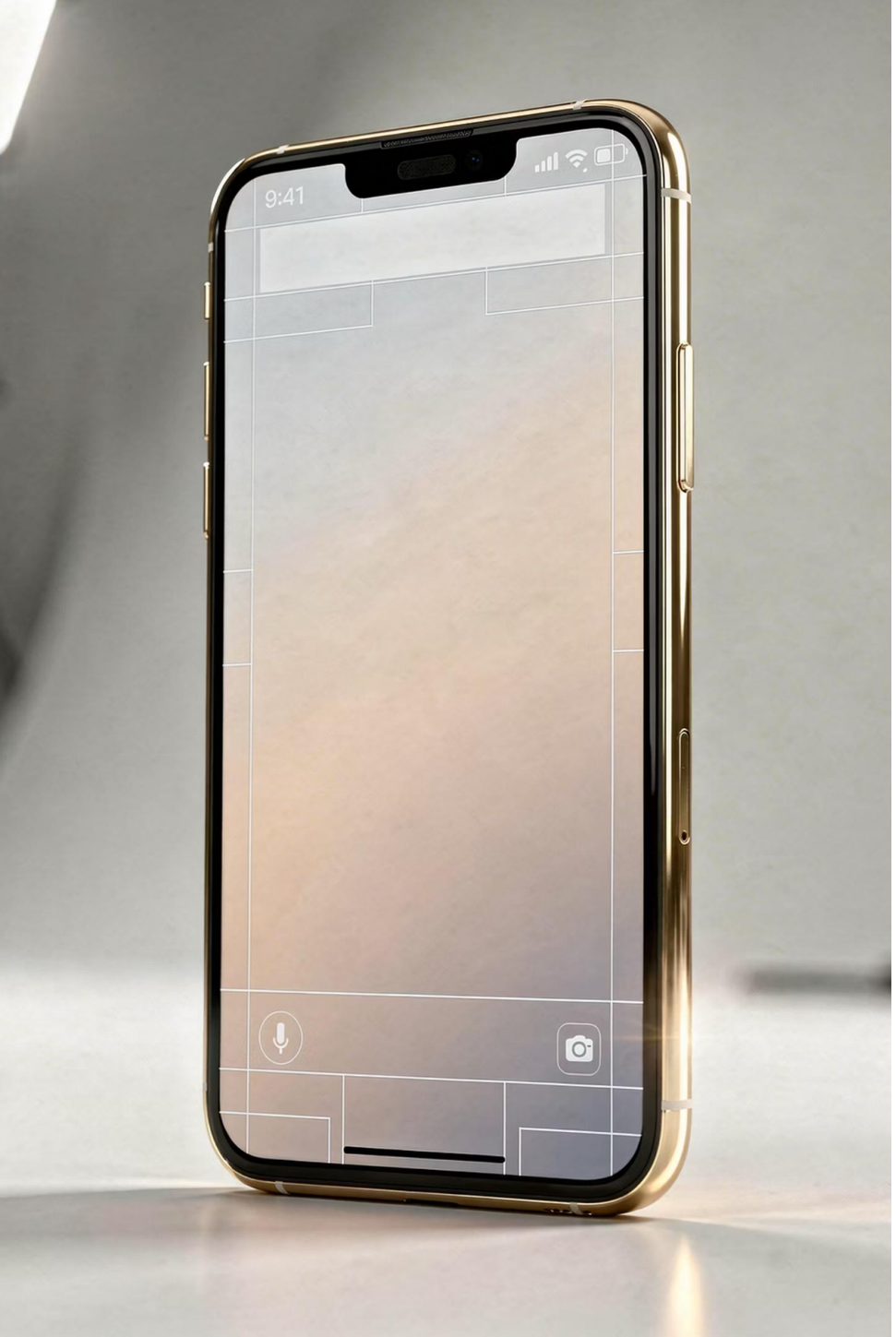
```
<TextInput
  value={text}
  onChangeText={setText}
  style={style.input}
  keyboardType="numeric"
/>
```

Experimente trocar "numeric" por "email-address", "phone-pad" ou "decimal-pad" para ver diferentes layouts.
OBS: Lembre-se que não é visivel pelo Browser.

02

Safe Area e Compatibilidade

Proteção de Conteúdo em Telas Modernas



O Desafio das Telas Modernas

Com a evolução dos smartphones, os fabricantes começaram a maximizar a área de exibição incorporando câmeras frontais diretamente na tela, criando o chamado "notch" (entalhe). Essa mudança de design trouxe um desafio significativo para desenvolvedores: como garantir que o conteúdo do aplicativo não fique oculto ou sobreposto por elementos físicos do dispositivo?

Além do notch, outros elementos podem interferir na visualização do conteúdo, como a barra de status no topo da tela, bordas arredondadas, e a barra de gestos na parte inferior dos dispositivos mais recentes. Cada fabricante e modelo pode ter configurações diferentes.

A Solução: Safe Area View

O Problema

- Conteúdo oculto pelo notch da câmera
- Sobreposição com barra de status
- Interferência com bordas arredondadas
- Conflito com barra de gestos inferior
- Variações entre diferentes modelos

A Solução

- Detecção automática de áreas seguras
- Ajuste dinâmico para cada dispositivo
- Compatibilidade iOS e Android
- Implementação simples e eficaz
- Prevenção de problemas visuais

Boa Prática: Sempre utilize SafeAreaView no container principal de suas telas, mesmo que o problema não seja visível no seu dispositivo de teste. Isso garante compatibilidade com a diversidade de dispositivos no mercado.

Implementando Safe Area Context

O React Native possui um componente SafeAreaView nativo, porém ele funciona apenas no iOS. Para garantir compatibilidade multiplataforma, utilizamos a biblioteca `react-native-safe-area-context`, que já vem pré-instalada nos projetos Expo.

Instalação

Verifique se a biblioteca está instalada:

```
npm list react-native-
safe-area-context
```

Se necessário, instale com:

```
npx expo install react-native-
safe-area-context
```

Implementação no Código

```
import { SafeAreaView }
from 'react-native-safe-area-context'

export default function RootLayout() {
 return (
 <SafeAreaView style={style.mainContainer}>
 {/* Seu conteúdo aqui */}
 </SafeAreaView>
 )
}
```

03

Configuração de Ícones

Identidade Visual do Aplicativo

A Importância do Ícone do Aplicativo

O ícone do aplicativo é frequentemente o primeiro ponto de contato visual entre o usuário e seu produto. Ele aparece na tela inicial do dispositivo, na loja de aplicativos, nas notificações e em diversos outros contextos. Um ícone bem projetado comunica instantaneamente a identidade e propósito do aplicativo.

No desenvolvimento com React Native e Expo, a configuração do ícone é gerenciada através do arquivo `app.json`, que contém todas as configurações principais do projeto. Este arquivo permite definir diferentes ícones para diferentes plataformas, garantindo que seu aplicativo tenha a melhor aparência em cada ambiente.

Configurando Ícones no app.json

O arquivo `app.json` é o centro de configuração do seu projeto Expo. Nele, você define não apenas o ícone, mas também nome do aplicativo, versão, orientação de tela e muitas outras propriedades importantes.

Ícone Geral

A propriedade `icon` no nível raiz define o ícone padrão usado quando não há especificação de plataforma.

```
"icon":
"./assets/images/check.
png"
```

iOS Adaptive Icon

No iOS, o ícone configurado na raiz é utilizado automaticamente. Certifique-se de usar imagens em alta resolução (1024x1024 recomendado).

Android Adaptive Icon

Android utiliza ícones adaptativos com foreground e background separados para criar efeitos visuais.

```
"android": {
  "adaptiveIcon": {
    "foregroundImage":
"./assets/images/check.
png",
    "backgroundColor":
"#ffffff"
  }
}
```

Favicon Web

Para aplicações web, configure o favicon que aparecerá na aba do navegador.

```
"web": {
 "favicon":
"./assets/images/check.
png"
}
```

Aplicando as Configurações

Estrutura do app.json

```
{
  "expo": {
    "name": "lista-de-tarefas",
    "slug": "lista-de-tarefas",
    "version": "1.0.0",
    "icon": "./assets/images/check.png",
    "ios": {
      "supportsTablet": true
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/images/check.png",
        "backgroundColor": "#ffffff"
      }
    },
    "web": {
      "favicon": "./assets/images/check.png"
    }
  }
}
```

Aplicando as Mudanças

Após modificar o arquivo `app.json`, é necessário reiniciar o servidor de desenvolvimento para que as alterações tenham efeito.

Passo a passo:

1. Pressione Ctrl+C no terminal para parar o servidor
2. Execute `npx expo start` novamente
3. Se estiver em dispositivo físico, feche completamente o app Expo Go
4. Reabra e conecte-se novamente ao servidor

> **Nota: Em alguns casos, especialmente em simuladores, pode ser necessário limpar o cache com `npx expo start -c`**

04

Splash Screen

Tela de Carregamento Profissional

O Que é uma Splash Screen?

A splash screen é a primeira tela que o usuário vê ao abrir seu aplicativo. Ela aparece enquanto o aplicativo está carregando recursos, inicializando serviços e preparando a interface. Uma splash screen bem projetada não é apenas funcional — ela contribui significativamente para a percepção de qualidade e profissionalismo do aplicativo.

No contexto do React Native com Expo, a splash screen é configurada através do arquivo `app.json` e pode incluir uma imagem estática, animações ou GIFs. O tempo de exibição é determinado automaticamente pelo Expo, que mantém a tela visível até que o aplicativo esteja completamente carregado e pronto para interação.

Configurando a Splash Screen

A configuração da splash screen é feita através do plugin `expo-splash-screen` no arquivo `app.json`. Este plugin oferece diversas opções de personalização para criar uma experiência de carregamento atraente.

image

Caminho para a imagem que será exibida. Recomenda-se usar imagens em alta resolução (pelo menos 1242x2436 para dispositivos modernos).

imageWidth

Define a largura da imagem em pixels. Útil para controlar o tamanho de exibição sem redimensionar o arquivo original.

resizeMode

Controla como a imagem se adapta à tela. Opções: `contain` (mantém proporção), `cover` (preenche tela) ou `native`.

backgroundColor

Cor de fundo da splash screen em formato hexadecimal. Deve harmonizar com a imagem escolhida.

Exemplo de Configuração

```
"plugins": [
  "expo-router",
  [
    "expo-splash-screen",
    {
      "image": "./assets/images/check.png",
      "imageWidth": 200,
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    }
  ]
]
```

Esta configuração cria uma splash screen com:

- Imagem centralizada de 200px de largura
- Modo contain para manter proporções
- Fundo branco limpo e profissional
- Carregamento automático gerenciado pelo Expo

Boas Práticas

- Use imagens vetoriais ou PNGs de alta qualidade
- Mantenha o design simples e clean
- Evite textos pequenos que podem ficar ilegíveis
- Teste em diferentes tamanhos de tela
- Considere o contraste entre imagem e fundo
- Alinhe o estilo com a identidade visual do app

Dica: Para aplicativos profissionais, considere criar animações Lottie ou GIFs sutis para tornar a espera mais agradável.

Agora que compreendemos todos os conceitos de tipos de teclado, safe area, ícones e splash screen, vamos revisar o que vimos do nosso aplicativo Lista de Tarefas com todas as implementações aplicadas.

- SafeAreaView Implementada

Substituímos o View container principal por SafeAreaView da biblioteca react-native-safe-area-context, garantindo compatibilidade com todos os dispositivos.

- Splash Screen Configurada

Adicionamos a splash screen com a mesma imagem do ícone, criando identidade visual consistente.

Tipo de Teclado Definido

- Ícone Personalizado

Configuramos o ícone check.png para todas as plataformas (iOS, Android e Web) através do app.json.

- **O TextInput está configurado com keyboardType="default", mas pode ser alterado conforme necessidade (numeric, email-address, etc).**

Próximos Passos: Na próxima aula, exploraremos o armazenamento local de dados no dispositivo, permitindo que as tarefas sejam salvas mesmo após fechar o aplicativo.

Estrutura Completa do Projeto

app/layout.jsx

Configuração da Stack com todas as rotas

app/index.jsx

Página inicial com links de navegação

app/products/index.jsx

Lista de produtos com links dinâmicos

app/products/[id].jsx

Detalhes do produto com parâmetro

app/+not-found.jsx

Página de erro personalizada

Principais Aprendizados

- Rotas aninhadas organizam a navegação em hierarquias lógicas usando pastas
- Arquivos `index.jsx` representam a rota raiz de cada pasta
- Colchetes `[nome]` criam parâmetros dinâmicos nas rotas
- Hook `useLocalSearchParams()` captura valores dos parâmetros
- Arquivo `+not-found.jsx` personaliza a página de erro 404
- Usar `router.replace()` evita retorno indesejado à página de erro

Firjan SENAI