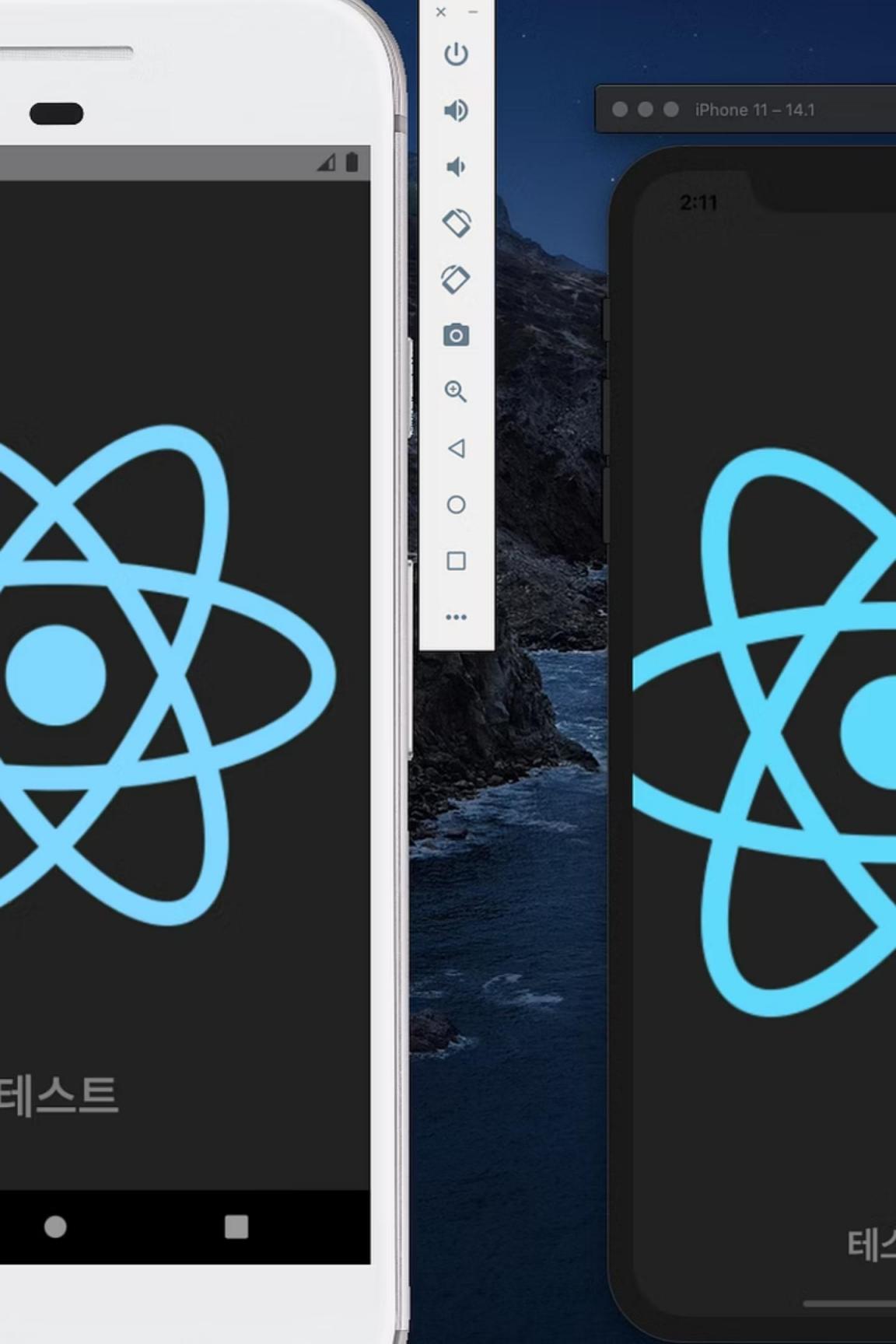


Desenvolvimento de Aplicativo Mobile

Material base desenvolvido pelo
Prof. Raphael Barreto

raphael.b.oliveira@docente.senai.br

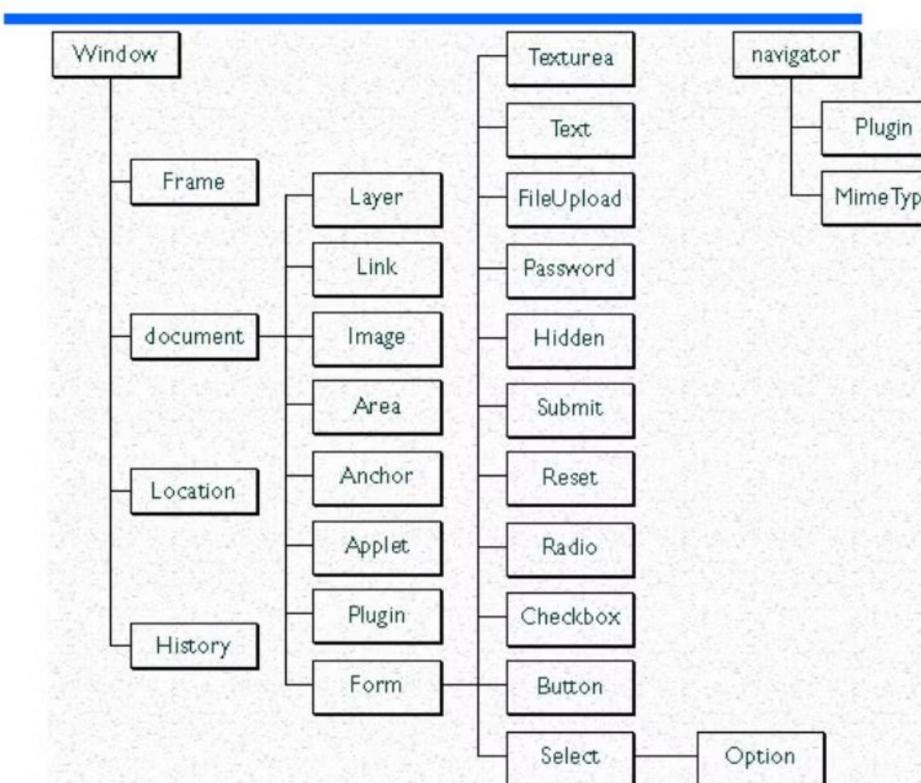


Do HTML ao React Native: Componentes Nativos

Entenda como funciona a interface visual em aplicativos móveis e faça a transição do desenvolvimento web para o mobile de forma natural e intuitiva.

O Desafio: Não Estamos Mais no Navegador

HTML DOM Structure



No React tradicional, usamos tags HTML como `div`, `button`, `p` e outras para construir interfaces. Essas tags funcionam perfeitamente no navegador porque temos acesso ao DOM.

Mas e no React Native?

Não existe navegador, não existe DOM, então não temos HTML! Como vamos criar nossas interfaces então?

É exatamente isso que vamos descobrir nesta aula.

A Solução: Core Components do React Native

O React Native resolve esse problema oferecendo seus próprios componentes visuais, chamados de **Core Components**. Eles são semelhantes ao HTML, mas adaptados para funcionar em dispositivos móveis nativos.

Importação Direta

Todos os componentes vêm direto do pacote React Native

Nativos de Verdade

Renderizam elementos nativos do iOS e Android

Sintaxe Familiar

Parecidos com HTML, mas com pequenas diferenças importantes

Onde Encontrar os Core Components

A documentação oficial do React Native tem uma seção dedicada aos **Core Components and APIs**. É lá que você vai encontrar todos os componentes disponíveis, suas propriedades e exemplos de uso.

A documentação é muito bem feita e traz snippets de código interativos para você experimentar. Recomendo fortemente que você a mantenha sempre aberta enquanto desenvolve!

[Core Components and APIs · React Native](#)

Basic Components

Most apps will end up using one or more of these basic components.

View

The most fundamental component for building a UI.

Text

A component for displaying text.

Image

A component for displaying images.

TextInput

A component for inputting text into the app via a keyboard.

Pressable

A wrapper component that can detect various stages of press interactions on any of its children.

ScrollView

Provides a scrolling container that can host multiple components and views.

StyleSheet

Provides an abstraction layer similar to CSS stylesheets.

Os Componentes Básicos Essenciais

Vamos conhecer os core components que você vai usar em praticamente todos os seus aplicativos. Esses são os blocos fundamentais de construção de qualquer interface mobile.

View

O equivalente à `div`. É o container básico para agrupar outros componentes.

Text

Substitui `p`, `h1`, `span` e todas as tags de texto. Todo texto precisa estar dentro dele.

Image

Para exibir imagens, similar ao `img` do HTML, mas com propriedades diferentes.

TextInput

Campo de entrada de texto, equivalente ao `input` do HTML.

Mais Componentes Essenciais

ScrollView

Uma `View` especial que permite rolagem quando o conteúdo ultrapassa o tamanho da tela. Perfeito para listas e conteúdo longo.

StyleSheet

A forma de adicionar "CSS" ao seu app. Vamos falar mais sobre ele na próxima aula, prometo!

Button

Um botão simples e básico. Funciona bem, mas tem limitações de personalização.

Switch

Renderiza uma entrada booleana.

⚠ Atenção

Armadilha Comum: Import Errado

Um dos erros mais comuns (e frustrantes!) é importar do lugar errado. Quando você digita o nome do componente, o autocomplete pode sugerir **React Native Web** ao invés de **React Native**.

- ❑ **Importante:** React Native Web é para rodar apps React Native no navegador como PWA. Se você importar dele, seu app vai quebrar no celular com erros estranhos sobre elementos HTML!

O Erro que Você Vai Ver (e Como Evitá-lo)

Quando você accidentalmente importa do React Native Web, recebe um erro misterioso dizendo que o app está tentando renderizar uma `div`, mas o celular não sabe o que é isso.

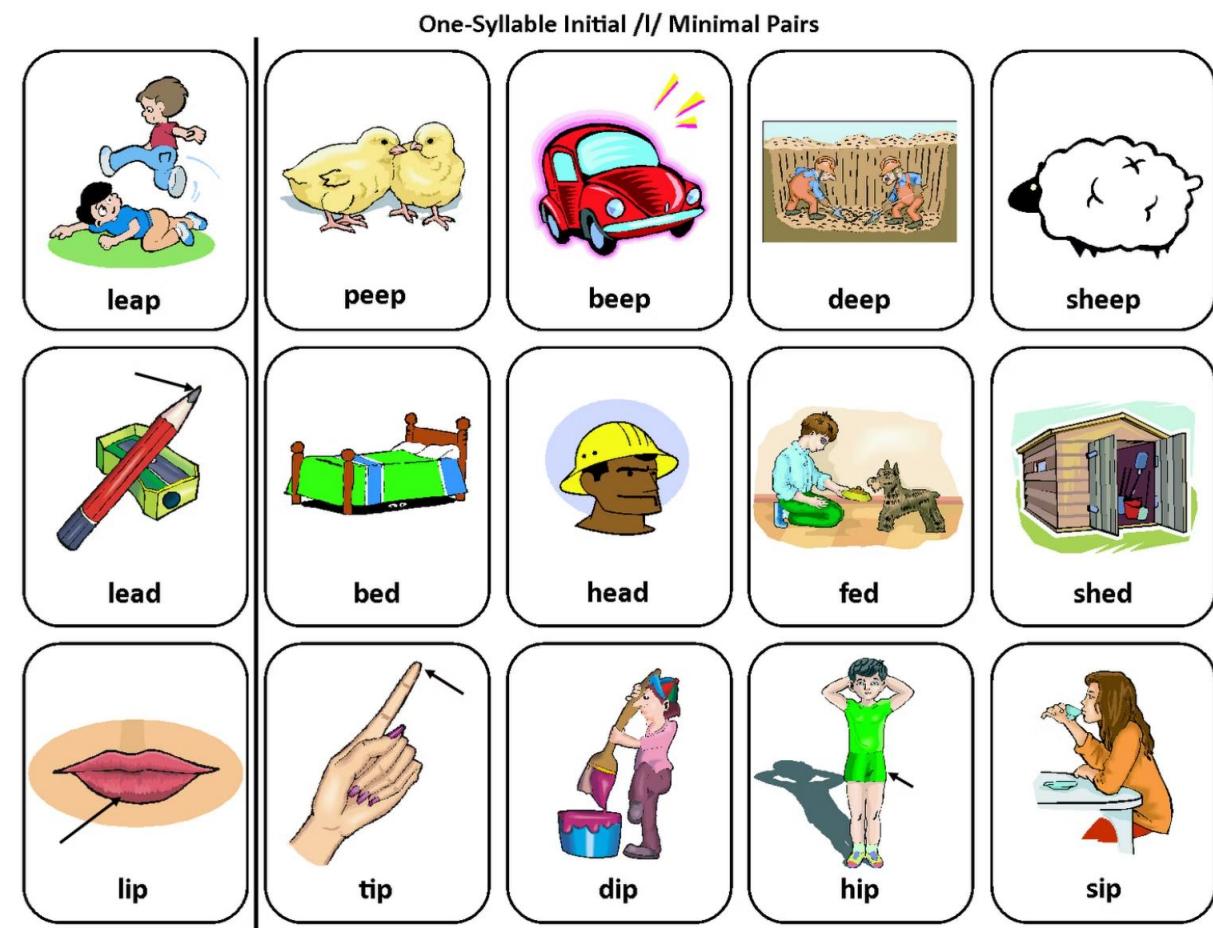
```
//
```

Vamos Construir uma To-Do List

Nosso Primeiro App Real

Para aprender esses conceitos na prática, vamos criar uma lista de tarefas simples. É o projeto perfeito para explorar os core components básicos e entender como eles funcionam juntos.

Ao longo do caminho, vamos descobrir as pequenas diferenças em relação ao HTML tradicional e como lidar com elas.



Começando com ScrollView

Vamos começar com a `ScrollView`, que vai permitir que nossa lista role quando tiver muitas tarefas. É como uma `div`, mas com superpoderes de rolagem!

```
import { ScrollView, Text } from 'react-native';

function App() {
  return (
    <ScrollView>
      <Text>Olá mundo!</Text>
      {/* Muito mais conteúdo aqui... */}
    </ScrollView>
  );
}
```

No simulador, você pode rolar com a rodinha do mouse. No celular real, vai funcionar naturalmente com o dedo. Mágico! ✨

Adicionando uma Imagem

Vamos adicionar um logo para nossa lista de tarefas. Primeiro, organizamos nossos assets em uma estrutura de pastas: `assets/images/`.

```
import { Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
    </ScrollView>
  );
}
```

Repare na primeira diferença importante: não é `src`, é `source`! Pequenas mudanças como essa são comuns ao migrar do HTML para React Native.

A imagem vai aparecer... gigante e sem proporção.

Calma!

Vamos estilizar isso na próxima aula com CSS.

Por enquanto, aceite a imagem caótica. 😊

Adicionando Texto

Todo texto em React Native precisa estar dentro de um componente Text. Não dá pra simplesmente jogar texto solto como no HTML!

```
import { Text, Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
      <text>Minhas tarefas</text>
    </ScrollView>
  );
}
```

O Text substitui todas as tags de texto do HTML: p, h1, h2, span, etc. Você vai estilizar depois para definir se é um título, parágrafo ou qualquer outra coisa.

Campo de Entrada: TextInput

Para permitir que o usuário digite novas tarefas, precisamos de um campo de entrada. O componente `TextInput` é o equivalente ao `input` do HTML.

```
import { TextInput, Text, Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
      <text>Minhas Tarefas</text>
      <TextInput placeholder="Adicione um item" />
    </ScrollView>
  );
}
```

Quando você clica nele no simulador ou toca no celular, o teclado aparece automaticamente. Magia! 

Só tem um problema: sem CSS, ele fica invisível - não tem borda nenhuma! Você só percebe que ele existe quando clica e o teclado aparece. Vamos resolver isso na próxima aula com estilização.

 INTERAÇÃO

A Grande Diferença: Não é Click, é Press!

No navegador, usamos `onClick` porque temos um mouse que *clica*. Mas em um celular, você não clica - você **pressiona** a tela com o dedo!

Por isso, no React Native o evento é `onPress`, não `onClick`. Parece pequeno, mas é uma diferença fundamental que reflete a natureza da interação mobile.

Criando um Botão Básico

Vamos adicionar um botão para adicionar novas tarefas à lista. O componente `Button` é bem direto, mas tem suas peculiaridades.

```
import { Button, TextInput, Text, Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
      <text>Minhas Tarefas</text>
      <TextInput placeholder="Adicione um item" />
      <Button title="Pressionar" onPress={() => alert("Botão pressionado!")} />
    </ScrollView>
  );
}
```

Diferente do HTML, o texto não vai entre as tags de abertura e fechamento. Você passa através da prop `title`. É uma das pequenas diferenças que você precisa se acostumar!

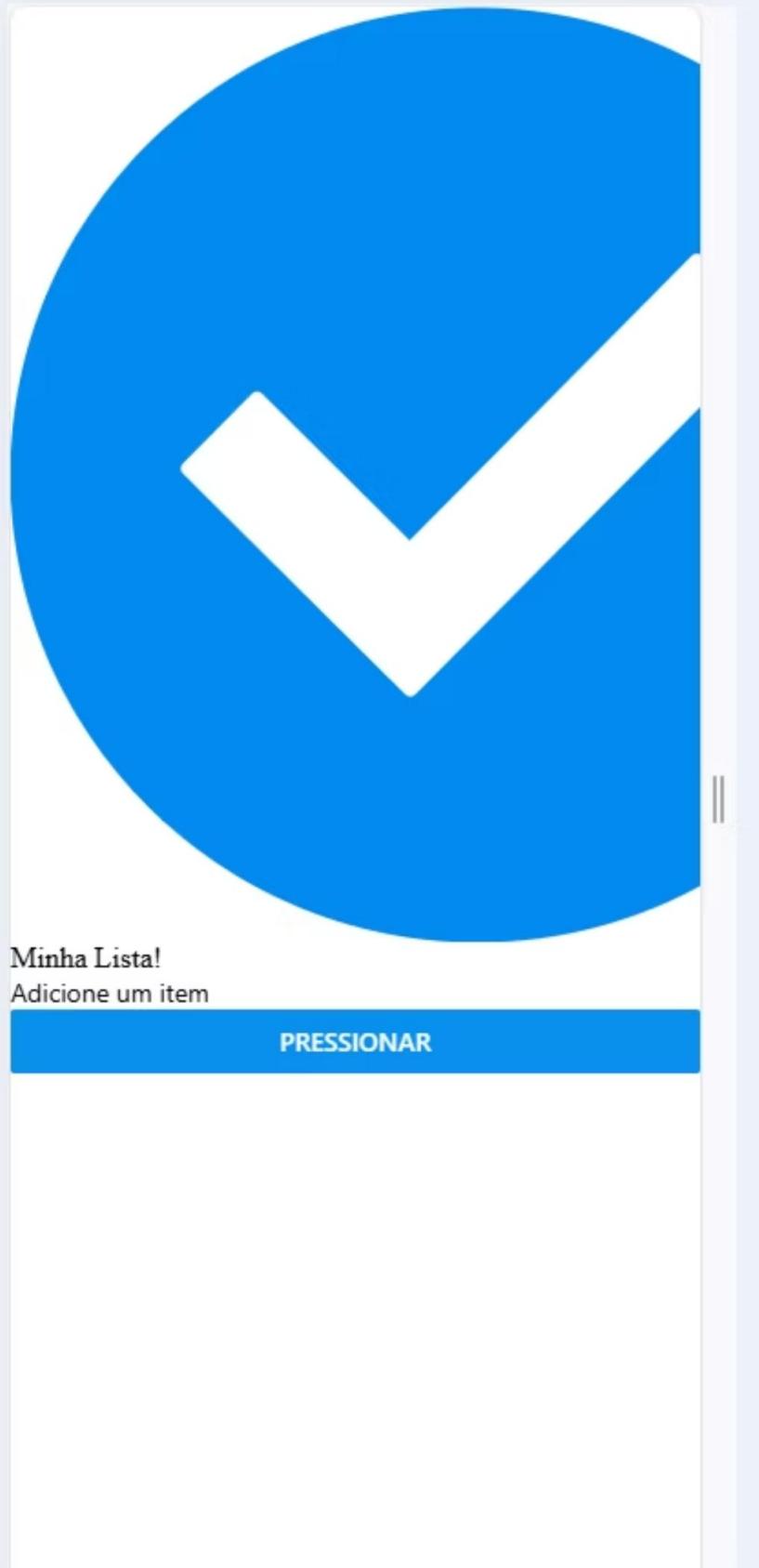
Mostrando Alertas no Mobile

Para testar nosso botão, vamos usar um alerta. Mas atenção: não é o `alert()` do navegador!

```
import { Alert, Button, TextInput, Text, Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
      <text>Minhas Tarefas</text>
      <TextInput placeholder="Adicione um item" />
      <Button title="Pressionar" onPress={() => Alert.alert("Botão pressionado!")} />
    </ScrollView>
  );
}
```

O `Alert` do React Native é uma classe, então você usa `Alert.alert()`. Se tentar usar só `Alert()` como função, vai dar erro dizendo que não pode executar uma classe. Contudo, se estiver usando o browser para visualizar o projeto, não será executado.



Nossa Interface Até Agora

Um Belo Desastre Visual

Sim, está feio. Muito feio. A imagem está gigante, o input é invisível, o botão está desalinhado... mas está **funcionando!**

Todos os componentes estão lá, fazendo o que deveriam fazer. A aparência horrível é temporária - mais a frente, vamos adicionar CSS e transformar isso em algo bonito.

COMPONENTE AVANÇADO

Button vs Pressable: Qual Usar?

O Button é simples e funcional, mas tem limitações. Você não consegue personalizá-lo muito - a cor, o título e pouco mais.

Button

Componente básico e limitado. Bom para protótipos rápidos, mas não para apps reais que precisam de design customizado.

Pressable

Componente moderno e totalmente customizável. A própria documentação recomenda usar este ao invés do Button.

Conhecendo o Pressable

O Pressable é a evolução do Button. Ele é totalmente estilizável e oferece vários tipos de eventos de toque para você trabalhar.

01

onPressIn

Dispara quando você **começa** a pressionar (dedo toca a tela)

02

onPressOut

Dispara quando você **solta** o dedo da tela

03

onPress

Dispara quando você completa o toque (pressionou e soltou)

04

onLongPress

Dispara quando você segura por mais de 500ms

[Pressable · React Native](#)

Pressable: Sintaxe e Uso

Diferente do `Button`, o `Pressable` aceita `children`, então você coloca o conteúdo entre as tags de abertura e fechamento - muito mais parecido com HTML!

```
import { Pressable, TextInput, Text, Image, ScrollView } from 'react-native';
import logo from "../assets/images/check.png";

export default function RootLayout() {
  return (
    <ScrollView>
      <Image source={logo} />
      <text>Minhas Tarefas</text>
      <TextInput placeholder="Adicione um item" />
      <Pressable onPress={() => alert("Pressionado!")}>
        <Text>Adicionar</Text>
      </Pressable>
    </ScrollView>
  );
}
```

Você pode colocar qualquer coisa dentro: texto, imagens, ícones, outros componentes... É muito mais flexível! Por isso vamos usar `Pressable` muito mais que `Button` ao longo do curso.

Outros Componentes Tocáveis (Não Use!)

A documentação também menciona `TouchableHighlight`, `TouchableOpacity` e `TouchableWithoutFeedback`. Mas ela mesma te avisa: **não use esses!**

`TouchableHighlight`

Clareia o componente ao tocar

`TouchableOpacity`

Deixa o componente opaco ao tocar

`TouchableWithoutFeedback`

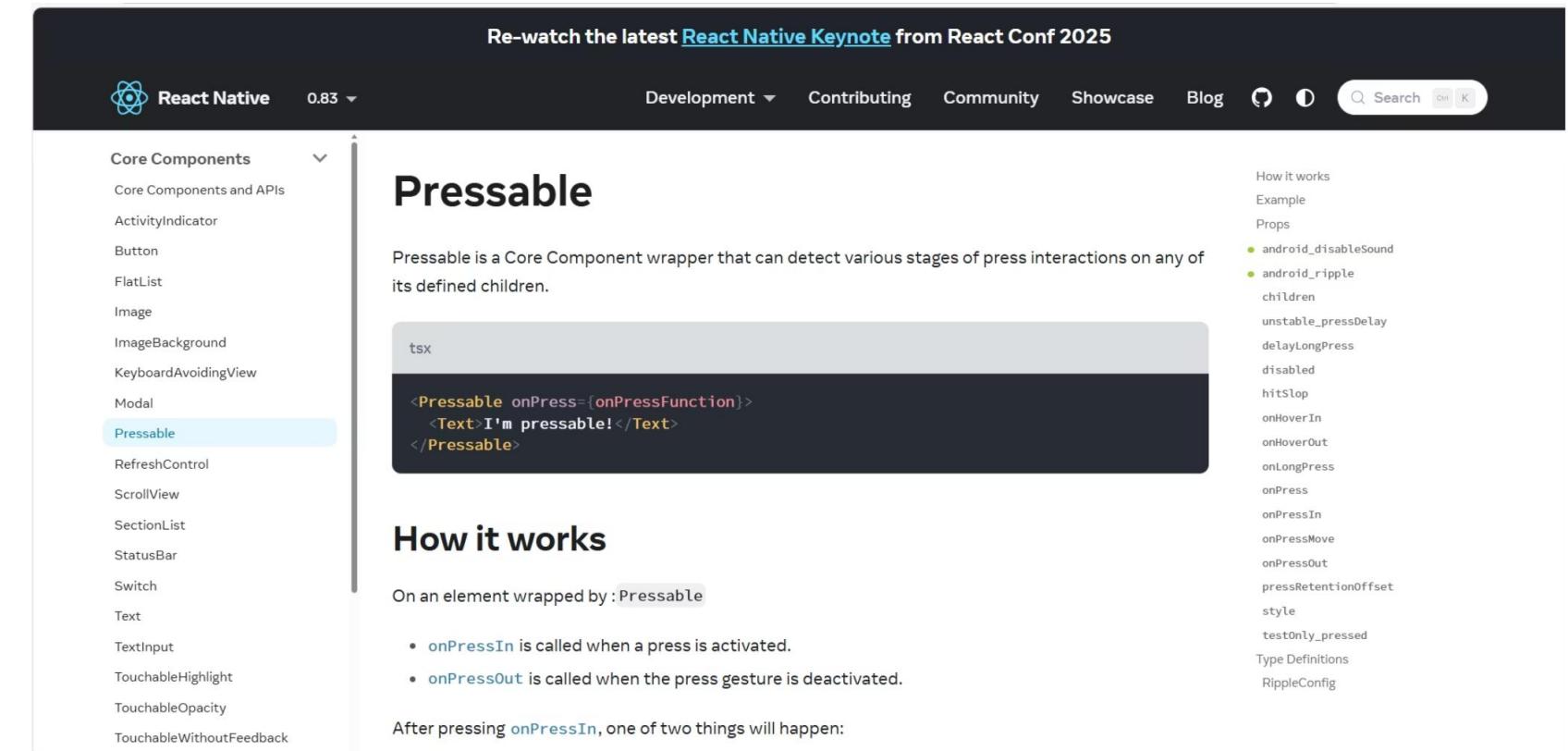
Sem feedback visual (não recomendado para UX!)

São componentes antigos. Use `Pressable` para tudo - ele é mais moderno, tem suporte garantido e você consegue criar os mesmos efeitos visuais com estilização.

Explorando a Documentação

A documentação do React Native é seu melhor amigo. Cada core component tem sua própria página com:

- Lista completa de props disponíveis
- Eventos que você pode usar
- Exemplos de código interativos
- Explicações detalhadas de comportamento
- Dicas de uso e boas práticas



The screenshot shows the React Native documentation page for the **Pressable** component. At the top, there's a navigation bar with the React Native logo, version 0.83, and links for Development, Contributing, Community, Showcase, and Blog. A search bar is also present. The main content area has a dark header with the title "Pressable". Below it, a paragraph explains that Pressable is a Core Component wrapper that can detect various stages of press interactions. A code snippet in a `tsx` block shows how to use it:

```
tsx
<Pressable onPress={onPressFunction}>
  <Text>I'm pressable!</Text>
</Pressable>
```

On the right side, there's a sidebar with links for "How it works", "Example", "Props", and a long list of prop names starting with "android_disableSound" and ending with "RippleConfig".

Pressable

Pressable is a Core Component wrapper that can detect various stages of press interactions on any of its defined children.

```
tsx
<Pressable onPress={onPressFunction}>
  <Text>I'm pressable!</Text>
</Pressable>
```

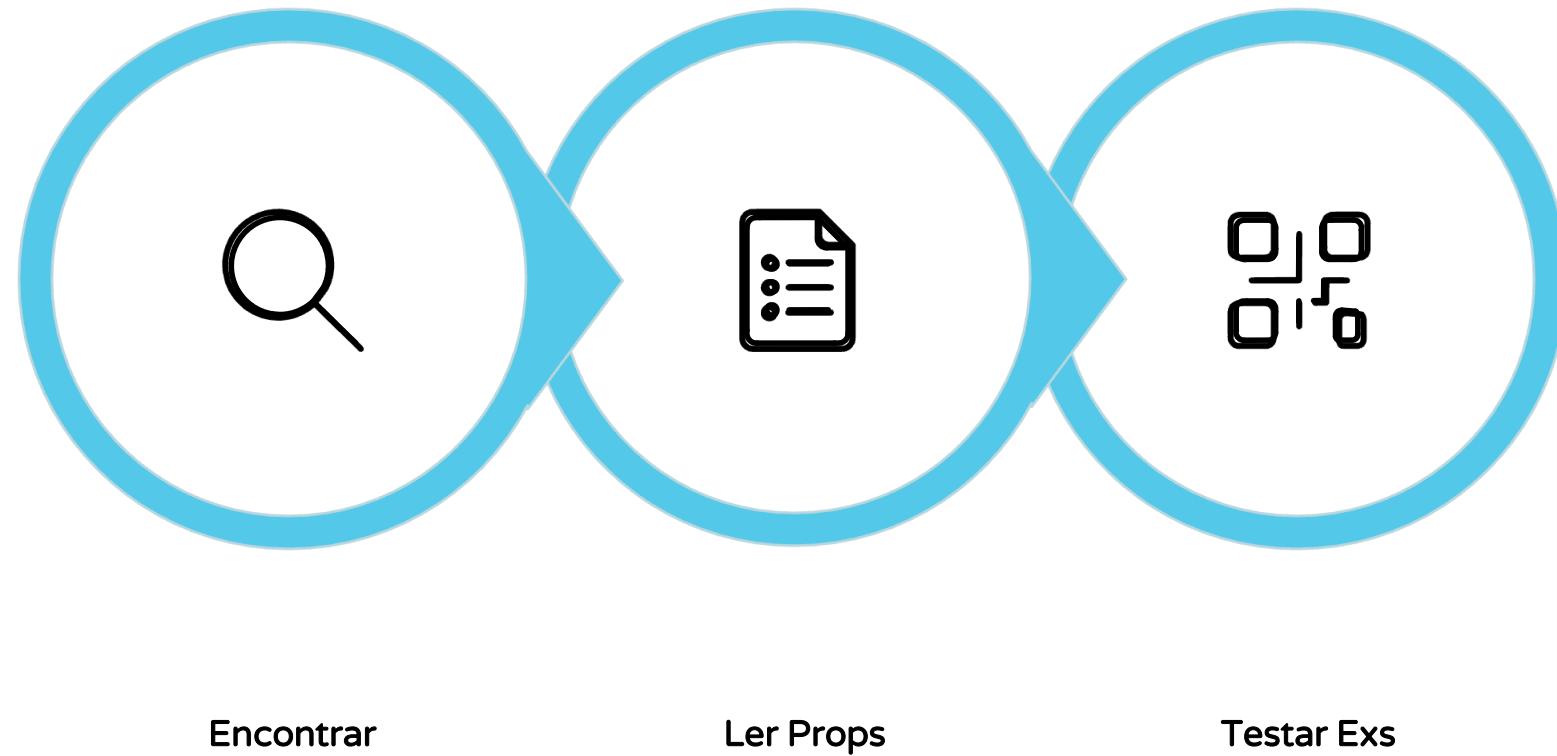
How it works

On an element wrapped by :Pressable

- `onPressIn` is called when a press is activated.
- `onPressOut` is called when the press gesture is deactivated.

After pressing `onPressIn`, one of two things will happen:

Como Usar a Documentação Eficientemente



Sempre que precisar usar um componente novo, siga esse fluxo: encontre o componente na seção Core Components, leia a lista de props e eventos disponíveis, e experimente os snippets de código interativos que a documentação oferece.

Exemplo Prático: Consultando Props

Digamos que você quer personalizar seu botão. Você vai até a documentação do `Button` e vê todas as props disponíveis na barra lateral direita.

`title`

O texto do botão

`onPress`

Função ao pressionar

`color`

Cor do botão

`accessibilityLabel`

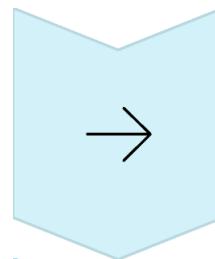
Para acessibilidade

`disabled`

Desabilitar o botão

E logo percebe: são poucas opções! Por isso o próprio React Native te direciona para o `Pressable` quando você precisa de mais controle.

Resumo das Principais Diferenças HTML → React Native



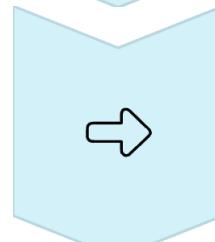
Tags Diferentes

`div` vira `View`, `p` vira `Text`, `img` vira `Image`



Props Diferentes

`src` vira `source`, `onClick` vira `onPress`



Import Obrigatório

Todos os componentes devem ser importados de `react-native`



Text Obrigatório

Todo texto precisa estar dentro de um componente `Text`

Dicas para Não Errar

1

Sempre confira o import

Certifique-se que está vindo de `react-native`, não `react-native-web`

2

Consulte a documentação

Antes de adivinhar o nome de uma prop, verifique na documentação oficial

3

Use Pressable, não Button

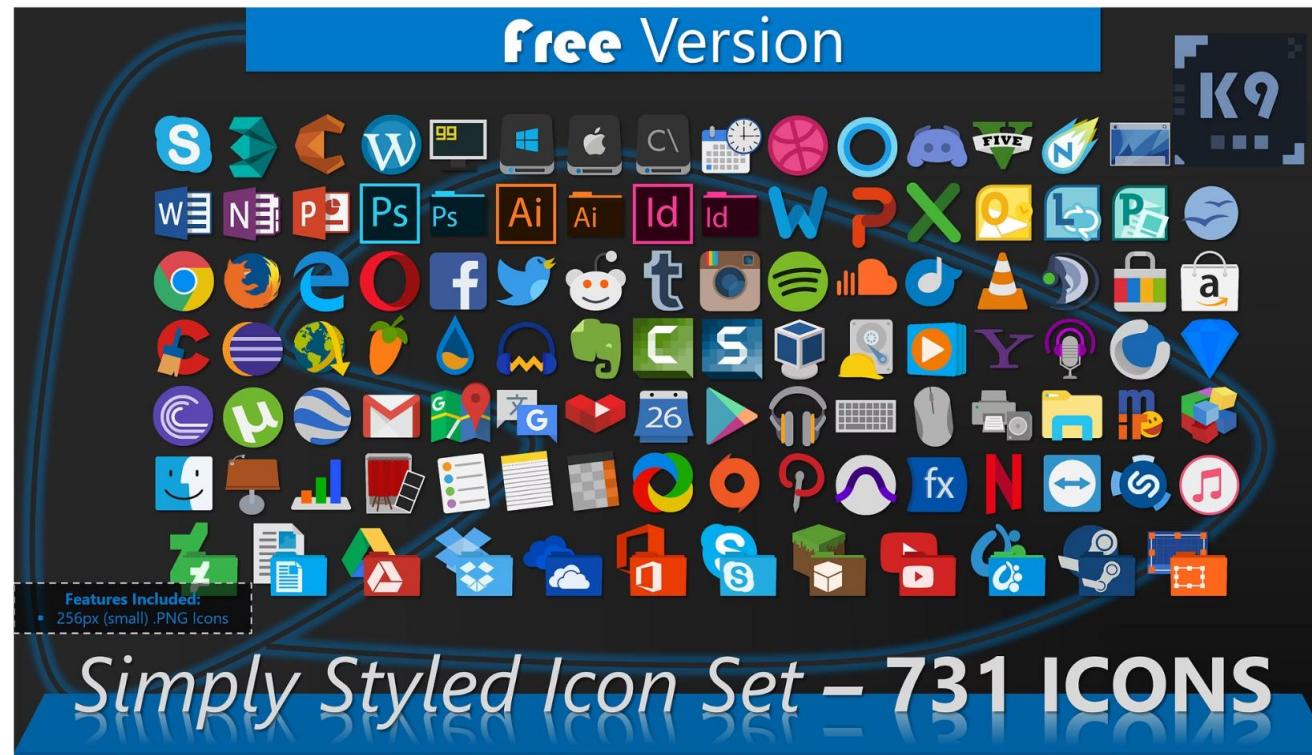
Para componentes customizáveis, prefira sempre `Pressable`

4

Lembre-se: é `onPress!`

No mobile não existe clique, existe pressão. Sempre `onPress`, nunca `onClick`

O Que Vem a Seguir



CSS Finalmente!

Na próxima aula, vamos transformar aquela interface horrível que criamos em algo bonito e profissional. Vamos aprender sobre `StyleSheet` e como estilizar componentes React Native.

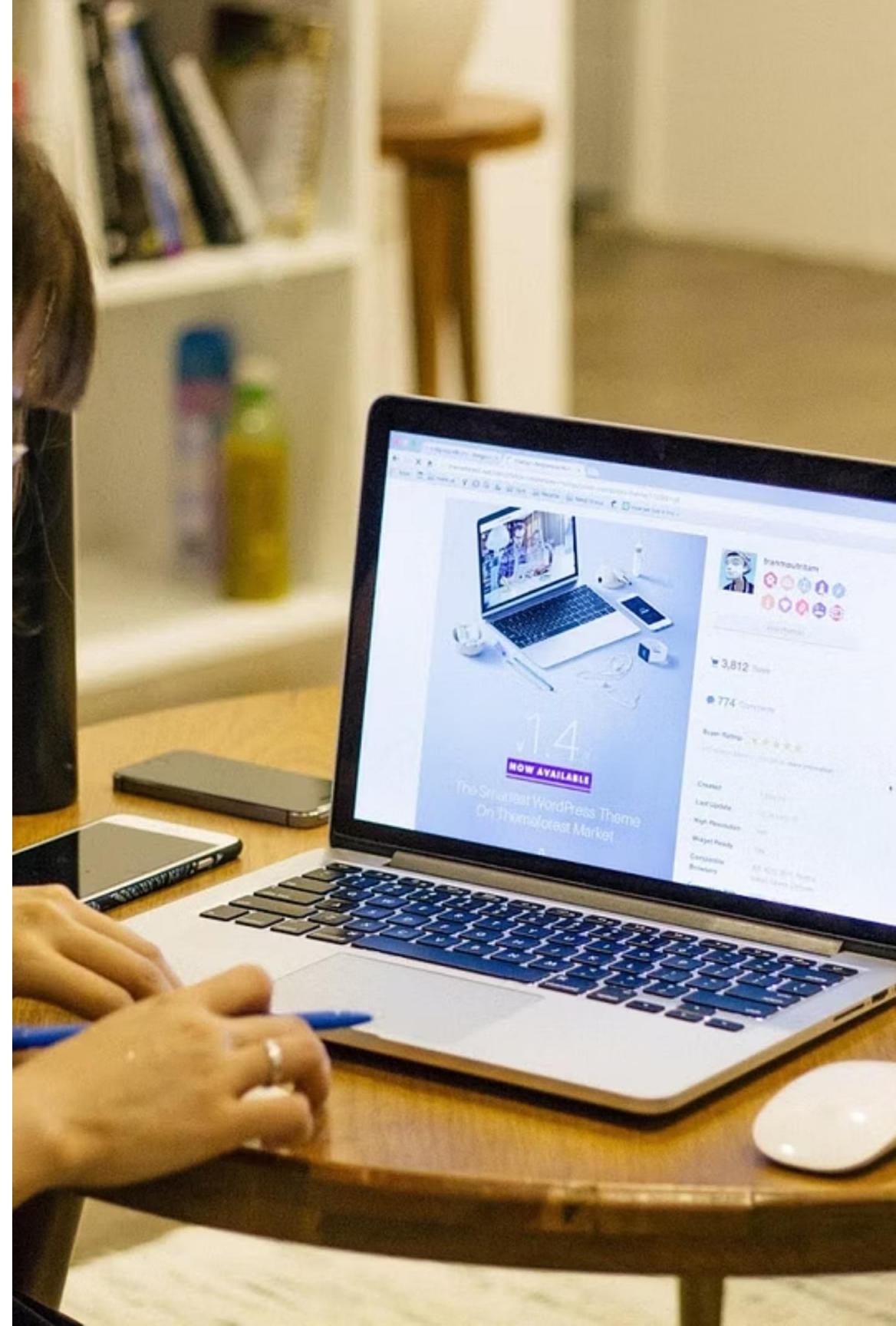
Você vai descobrir que o CSS no React Native é bem diferente do CSS tradicional, mas igualmente poderoso. Prepare-se para deixar seu app lindo! ♦♦

Você Está Pronto para o Mobile!



Parabéns! Você deu o primeiro passo importante na transição do web para mobile. Agora você entende a base de como os componentes funcionam no React Native e está preparado para criar interfaces nativas reais.

Na próxima aula, vamos adicionar aquele toque final que falta: estilização! Prepare-se para transformar código em design. Até lá! A small hand icon with fingers slightly spread, pointing towards the text.





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

