Interação com Componentes e APIs Nativos

Prof. Karan Luciano

Desenvolvimento Mobile - Ensino Médio

25 de outubro de 2024

Sumário

Objetivos da Apresentação

- Compreender o que são componentes nativos no React Native.
- Entender a diferença entre componentes nativos e componentes web tradicionais.
- Explorar componentes interativos no React Native.
- Aprender a utilizar APIs nativas como Alert e Clipboard.
- Implementar um exemplo prático consumindo uma API pública.

O que são Componentes Nativos?

- Componentes que mapeiam diretamente para elementos da interface nativa do dispositivo.
- Proporcionam desempenho e aparência nativa às aplicações.
- Exemplos: `View`, `Text`, `TextInput`.

Por que não usar Componentes Web?

- React Native não utiliza o DOM do navegador; não há suporte para tags HTML como `<div>`, `<span>`, `<input>`.
- Componentes web não interagem com as APIs nativas do dispositivo.
- Uso de componentes nativos garante melhor integração com funcionalidades do sistema operacional.

TextInput vs. input HTML

- `TextInput` é o componente nativo para entrada de texto no React Native.
- Mapeia para `UITextField` no iOS e `EditText` no Android.
- Oferece melhor controle sobre o teclado virtual e interação do usuário.

Visão Geral dos Componentes Interativos

- Componentes que respondem às ações do usuário.
- Facilitam a criação de interfaces dinâmicas.
- Exemplos: `ScrollView`, `TextInput`, `Button`.

ScrollView

- Container que permite rolagem de conteúdo.
- Suporta rolagem vertical e horizontal.
- Útil para listas pequenas ou conteúdos que excedem o tamanho da tela.

Exemplo de ScrollView

```
1  import React from 'react';
2  import { ScrollView, Text } from 'react-native';
3
4  const ScrollViewExample = () => (
5      <ScrollView>
6          {Array.from({ length: 30 }, (_, i) => (
7              <Text key={i} style={{ fontSize: 20 }}>
8                  Item {i + 1}
9              </Text>
10         ))}
11     </ScrollView>
12 );
13
14 export default ScrollViewExample;
15
```

TextInput

- Componente para entrada de texto pelo usuário.
- Suporta diversos tipos de teclado.
- Permite personalizar estilos e comportamentos.

Exemplo de TextInput

```
import React, { useState } from 'react';
import { View, TextInput, Text } from 'react-native';

const TextInputExample = () => {
    const [name, setName] = useState('');

    return (
        <View style={{ padding: 10 }}>
            <TextInput
                placeholder="Digite seu nome"
                onChangeText={setName}
                value={name}
                style={{
                    height: 40,
                    borderColor: 'gray',
                    borderWidth: 1,
                    marginBottom: 10,
                }}
            />
            <Text>Bem-vindo, {name}!</Text>
        </View>
```

Button

- Componente para capturar toques do usuário.
- Simples de usar, mas com opções de personalização limitadas.
- Para estilos personalizados, utilize `TouchableOpacity`.

Exemplo de Button

```
1  import React from 'react';
2  import { View, Button, Alert } from 'react-native';
3
4  const ButtonExample = () => (
5      <View style={{ padding: 10 }}>
6          <Button
7              title="Pressione-me"
8              onPress={() => Alert.alert('Bot o
       pressionado!')}
9          />
10     </View>
11 );
12
13 export default ButtonExample;
14
```

APIs Nativas

- O React Native fornece acesso a diversas APIs nativas do dispositivo.
- Permite utilizar funcionalidades como câmera, GPS, notificações, etc.
- Facilita a criação de aplicativos ricos em recursos.

Alert

- API para exibir diálogos de alerta ao usuário.
- Suporta múltiplos botões e configurações de estilo.
- Integra-se com o sistema operacional para aparência nativa.

Exemplo de Alert

```
import { Alert } from 'react-native';

const showAlert = () => {
    Alert.alert(
        'T tulo do Alerta',
        'Mensagem do alerta',
        [
            { text: 'Cancelar', style: 'cancel' },
            { text: 'OK', onPress: () => console.log('OK Pressionado') },
        ],
        { cancelable: false }
    );
};

```

Clipboard

- API para copiar e colar texto no clipboard do dispositivo.
- Útil para funcionalidades de compartilhamento e transferência de dados.

Exemplo de Clipboard

```
1  import React, { useState } from 'react';
2  import { View, Button, Text, Clipboard } from 'react-
       native';
3
4  const ClipboardExample = () => {
5      const [copiedText, setCopiedText] = useState('');
6
7      const copyToClipboard = () => {
8          Clipboard.setString('Texto para copiar');
9          Alert.alert('Texto copiado para o clipboard!');
10     };
11
12     const fetchCopiedText = async () => {
13         const text = await Clipboard.getString();
14         setCopiedText(text);
15     };
16
17     return (
18         <View style={{ padding: 10 }}>
19             <Button title="Copiar Texto" onPress={
       copyToClipboard} />
```

O que são Assets?

- Recursos estáticos utilizados na aplicação, como imagens, fontes e sons.
- Armazenados na pasta `assets` do projeto.
- Devem ser gerenciados para garantir performance e organização.

Importando Imagens Locais

- Use `require` para importar imagens locais.
- As imagens serão incluídas no bundle da aplicação.

Exemplo de Imagem Local

```
1  import React from 'react';
2  import { Image } from 'react-native';
3
4  const LocalImageExample = () => (
5      <Image
6          source={require('./assets/logo.png')}
7          style={{ width: 100, height: 100 }}
8      />
9  );
10
11 export default LocalImageExample;
12
```

Importando Imagens Remotas

- Utilize uma URI para carregar imagens da internet.
- Certifique-se de especificar dimensões para a imagem.

Exemplo de Imagem Remota

```
1  import React from 'react';
2  import { Image } from 'react-native';
3
4  const RemoteImageExample = () => (
5      <Image
6          source={{ uri: 'https://example.com/image.png' }}
7          style={{ width: 200, height: 200 }}
8      />
9  );
10
11 export default RemoteImageExample;
12
```

Escolha da API

- Utilizaremos a **PokéAPI** (https://pokeapi.co/).
- Fornece dados sobre Pokémon, incluindo nomes, imagens e estatísticas.
- Ideal para demonstrar consumo de APIs REST.

Objetivo do Exemplo

- Criar uma aplicação que lista Pokémon com nome e imagem.
- Utilizar componentes interativos como `ScrollView`.
- Demonstrar uso de `fetch`, `useEffect` e gerenciamento de estado.

Estrutura do Código

- Configurar estado inicial para armazenar dados dos Pokémon.
- Fazer requisição à API ao montar o componente.
- Atualizar o estado com os dados recebidos.
- Renderizar a lista de Pokémon com nome e imagem.

Configurando o Estado e useEffect

```
1  import React, { useState, useEffect } from 'react';
2
3  const PokemonList = () => {
4      const [pokemon, setPokemon] = useState([]);
5
6      useEffect(() => {
7          // Fun   o para buscar dados da API
8      }, []);
9
10     // Renderiza   o ser   adicionada posteriormente
11 };
12
13 export default PokemonList;
14
```

Fazendo a Requisição à API

```
1  useEffect(() => {
2      const fetchPokemon = async () => {
3          try {
4              const response = await fetch(
5                  'https://pokeapi.co/api/v2/pokemon?limit
       =10'
6              );
7              const data = await response.json();
8              setPokemon(data.results);
9          } catch (error) {
10             console.error(error);
11         }
12     };
13
14     fetchPokemon();
15 }, []);
16
```

Obtendo Detalhes de Cada Pokémon

- Para cada Pokémon, faremos uma requisição adicional para obter a imagem.

```
useEffect(() => {
    const fetchPokemon = async () => {
        try {
            const response = await fetch(
                'https://pokeapi.co/api/v2/pokemon?limit=10'
            );
            const data = await response.json();
            const detailedPokemon = await Promise.all(
                data.results.map(async (poke) => {
                    const res = await fetch(poke.url);
                    const details = await res.json();
                    return {
                        name: poke.name,
                        image: details.sprites.front_default,
                    };
                })
```

Renderizando a Lista de Pokémon (Parte 1)

```
1  import { ScrollView, View, Text, Image } from 'react-
       native';
2
3  // Dentro do componente PokemonList
4  return (
5      <ScrollView>
6          {pokemon.map((poke, index) => (
7              <View key={index} style={{ alignItems: '
       center', margin: 10 }}>
8                  <Text style={{ fontSize: 18,
       textTransform: 'capitalize' }}>
9                      {poke.name}
10                 </Text>
11                 {/* Imagem ser  adicionada na pr xima
       parte */}
12             </View>
13         ))}
14     </ScrollView>
15 );
16
```

Renderizando a Lista de Pokémon (Parte 2)

```
<Image
    source={{ uri: poke.image }}
    style={{ width: 100, height: 100 }}
/>

```

- Adicione o código acima dentro do `<View>` para exibir a imagem.

Considerações Finais sobre o Exemplo

- Demonstramos como consumir uma API REST no React Native.
- Utilizamos componentes nativos e interativos.
- O uso de `async/await` simplifica o código assíncrono.

Recapitulando

- Entendemos a importância dos componentes nativos no React Native.
- Exploramos componentes interativos como `ScrollView`, `TextInput` e `Button`.
- Aprendemos a utilizar APIs nativas como `Alert` e `Clipboard`.
- Implementamos um exemplo prático consumindo a PokéAPI.

Próximos Passos

- Experimentar outros componentes nativos e interativos.
- Integrar mais funcionalidades nativas em seus projetos.
- Explorar outras APIs públicas para ampliar suas habilidades.

Referências

- Documentação Oficial do React Native - Componentes e APIs
- PokéAPI - API de Pokémon
- React Native - TextInput

Perguntas?

Obrigado pela atenção!
Alguma pergunta?