

Vitor Amadeu Souza

Introdução ao

React Native

No ambiente Expo Snack

Parte XVII

© 2024 by Cerne Tecnologia e Treinamento Ltda.

© 2024 by Vitor Amadeu Souza

Nenhuma parte desta publicação poderá ser reproduzida sem autorização prévia e escrita de **Cerne Tecnologia e Treinamento Ltda.** Este livro publica nomes comerciais e marcas registradas de produtos pertencentes a diversas companhias. O editor utiliza as marcas somente para fins editoriais e em benefício dos proprietários das marcas, sem nenhuma intenção de atingir seus direitos.

Outubro de 2024

Direitos reservados por:

Cerne Tecnologia e Treinamento Ltda

Produção: Cerne Tecnologia e Treinamento

E-mail da Empresa: cerne@cerne-tec.com.br

Home Page: www.cerne-tec.com.br.com.br

Atendimento ao Consumidor: sac@cerne-tec.com.br

Contato com o Autor: vitor@cerne-tec.com.br



FEITO NO BRASIL

***“Venha também sobre mim a tua benignidade, ó Senhor, e a tua
salvação, segundo a tua palavra.”***

Sl 119:41

Cerne Tecnologia

A Cerne Tecnologia tem uma equipe preparada para desenvolvimento de projetos eletrônicos em diversas áreas: Médica, Entretenimento, Industrial, Robótica, Científica, Automobilística, Aeronáutica, etc. Trabalhamos com tecnologia microcontrolada usando o PIC, ARM, AVR, 8051, dsPIC, PIC24, PIC32 além do Arduino, Raspberry, Beaglebone etc. Desenvolvemos o projeto desde sua concepção até a entrega do produto final, passando pelas etapas de esquema elétrico, protótipo e desenvolvimento de circuito impresso.

Desenvolvemos aplicativos para smartphones/tablets Android, iOS, Blackberry, Windows Phone e no desenvolvimento de softwares a nível PC para plataforma Windows, usando ferramentas como o Visual Basic, C# e C++.

Atuamos na parte de montagem de placas, onde podemos fornecer ambos os serviços de desenvolvimento de projetos e produção ou apenas um destes.

Desenvolvemos esquemas elétricos e layout de PCB, tanto em tecnologia convencional como SMD.

Temos a flexibilidade de customizar um de nossos produtos, de modo a atender a uma necessidade específica do cliente, tornando o custo de desenvolvimento menor se comparado a construção de um projeto desde a sua fase inicial.

Desenvolvemos e fornecemos kits didáticos para diversos microcontroladores além de apostilas, livros e e-books.

Na hora de desenvolver um projeto ou equipar seu laboratório não hesite em nos contatar. Entre em contato conosco através do endereço cerne-tec.com.br para obter mais informações.



cerne-tec.com.br

Sumário

Capítulo I – Metodologia de desenvolvimento.....	6
1. Introdução.....	6
Capítulo II – Programação em React Native.....	7
1. Leitura de JSON.....	7
2. Módulo de estilo.....	12
3. Cotação de criptomoedas.....	15
4. Gráfico de Bitcoin.....	18

Capítulo I

Metodologia de desenvolvimento

1. Introdução

Neste livro, você encontrará uma parte da série dedicada ao desenvolvimento de projetos com React Native. Ao longo desta obra, diversos experimentos serão apresentados conforme o sumário, permitindo um aperfeiçoamento contínuo no uso deste framework para a construção de interfaces responsivas com foco em dispositivos móveis.

Capítulo II

Programação em React Native

1. Leitura de JSON

O código apresentado é um componente React Native chamado `RomanEmperors`, que tem como objetivo exibir uma lista de imperadores romanos, cada um acompanhado de uma imagem e seu nome. O componente utiliza hooks do React, como `useState` e `useEffect`, para gerenciar o estado e carregar dados de um arquivo JSON.

No início, o componente importa as bibliotecas necessárias do React e do React Native, incluindo elementos como `View`, `Text`, `Image`, `StyleSheet` e `FlatList`. Além disso, ele importa os dados de um arquivo chamado `data.json`, que contém informações sobre os imperadores.

O estado inicial `emperors` é configurado como um array vazio, utilizando o hook `useState`. Esse estado será utilizado para armazenar os dados dos imperadores. O hook `useEffect` é empregado para carregar os dados do arquivo JSON assim que o componente é

montado. Dentro desse hook, a função `setEmperors(data)` é chamada para atualizar o estado com os dados carregados.

Para renderizar cada item da lista, a função `renderCard` é definida. Essa função recebe um objeto `item`, que representa um imperador, e cria uma `View` contendo uma imagem (usando a URL fornecida no objeto) e o nome do imperador. O componente utiliza o `FlatList`, que é uma maneira eficiente de renderizar listas em `React Native`. O `FlatList` recebe os dados dos imperadores, a função de renderização (`renderCard`) e um método para gerar chaves exclusivas (`keyExtractor`). Neste caso, a chave é gerada a partir do índice do item na lista.

Os estilos são definidos utilizando o objeto `StyleSheet`. O contêiner principal tem um fundo cinza claro e `padding`, enquanto cada cartão que exibe um imperador possui fundo branco, bordas arredondadas, `padding`, sombra e elevação para criar um efeito de profundidade. As imagens dos imperadores são circulares, com tamanho definido e margem inferior para separar visualmente o texto.

Desta forma, esse componente apresenta uma forma eficaz de mostrar uma lista visualmente agradável de imperadores romanos,

utilizando imagens e texto. A abordagem modular com hooks permite que o código seja limpo e fácil de entender, enquanto o uso do FlatList garante desempenho otimizado ao lidar com listas potencialmente longas. Assim, o componente não apenas exibe informações relevantes, mas também proporciona uma experiência visual atraente ao usuário.

```
import React, { useEffect, useState } from 'react';
import { View, Text, Image, StyleSheet, FlatList } from 'react-native';
import data from './data.json';

export default function RomanEmperors() {
  const [emperors, setEmperors] = useState([]);

  useEffect(() => {
    // Carrega os dados do arquivo JSON
    setEmperors(data);
  }, []);

  const renderCard = ({ item }) => (
    <View style={styles.card}>
      <Image source={{ uri: item.image }} style={styles.image} />
      <Text style={styles.name}>{item.name}</Text>
    </View>
  );

  return (
    <View style={styles.container}>
      <FlatList
        data={emperors}
        renderItem={renderCard}
        keyExtractor={(item, index) => index.toString()}
      />
    </View>
  );
}

const styles = StyleSheet.create({
```

```

container: {
  flex: 1,
  backgroundColor: '#f0f0f0',
  padding: 10,
},
card: {
  backgroundColor: '#fff',
  borderRadius: 10,
  padding: 20,
  marginVertical: 10,
  alignItems: 'center',
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.3,
  shadowRadius: 5,
  elevation: 5,
},
image: {
  width: 100,
  height: 100,
  borderRadius: 50,
  marginBottom: 10,
},
name: {
  fontSize: 18,
  fontWeight: 'bold',
},
});

```