Vitor Amadeu Souza

Introdução ao

React Native

No ambiente Expo Snack

Parte XVIII

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 PCI, 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
Conhecimento para o Desenvolvimento

cerne-tec.com.br

Sumário

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. Tela de Checkout

Esse código implementa um aplicativo React Native que simula um fluxo de checkout e envio de confirmação de pedido por e-mail. Ele é dividido em três componentes principais: App.js, Checkout.js e CheckoutScreen.js. O arquivo App.js configura a navegação principal do aplicativo, utilizando NavigationContainer e createStackNavigator para definir uma pilha de navegação com uma tela inicial chamada "Checkout", que exibe o componente CheckoutScreen. No arquivo Checkout.js, o usuário encontra um formulário de checkout para inserir dados como nome, endereço e informações do pedido. O formulário utiliza o useState para armazenar cada campo de entrada e inclui uma função de validação para garantir que todos os campos estejam preenchidos antes de permitir a finalização do pedido. Os produtos, quantidades e preços estão pré-definidos, com um cálculo total feito com base nas quantidades e preços fixados no código. Ao pressionar o botão "Finalizar Pedido", a função finalizarPedido verifica a validade dos campos e exibe uma mensagem de confirmação.

No arquivo CheckoutScreen.js, o usuário insere dados adicionais, como rua, cidade e CEP, e pode enviar um e-mail com o resumo do pedido. Para isso, o código utiliza o expo-mail-composer, que oferece uma função para enviar e-mails com os dados preenchidos. A função MailComposer.composeAsync() permite que o e-mail seja enviado para o endereço configurado no campo recipients, e uma mensagem de erro ou sucesso é exibida ao usuário conforme o resultado. Além disso, o botão "Limpar Tudo" redefine os campos do formulário para os valores iniciais. Essa estrutura é facilmente expansível para incluir novas telas, adicionar produtos ou modificar as funcionalidades, como ajustar o endereço de e-mail destinatário para envio do pedido.

```
//App.js
import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import CheckoutScreen from './CheckoutScreen';

const Stack = createStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Checkout">
        <Stack.Screen name="Checkout" component={CheckoutScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

```
//Checkout.js
import React, { useState } from 'react';
import {View,Text,StyleSheet,KeyboardAvoidingView} from 'react-
native';
import { TextInput, Button } from 'react-native-paper';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';


const Stack = createStackNavigator();


const Checkout = () => {
  const [nome, setNome] = useState('');
  const [endereco, setEndereco] = useState('');
  const [cidade, setCidade] = useState('');
  const [estado, setEstado] = useState('');
  const [cep, setCep] = useState('');
  const [telefone, setTelefone] = useState('');
  const [email, setEmail] = useState('');
  const [pedido, setPedido] = useState({
    produtos: [
      { nome: 'Produto 1', quantidade: 2, preco: 10.99 },
      { nome: 'Produto 2', quantidade: 1, preco: 5.99 },
    ],
  });


  const validarCampos = () => {
    if (!nome||!endereco||!cidade||!estado||!cep||!telefone||!email) {
      alert('Preencha todos os campos!');
      return false;
    }
    return true;
  };


  const finalizarPedido = () => {
    if (validarCampos()) {
      // Enviar pedido para o servidor
      console.log('Pedido finalizado!');
    }
  };


  return (
    <KeyboardAvoidingView behavior="padding" style={styles.container}>
      <Text style={styles.titulo}>Checkout</Text>
      <View style={styles.form}>
        <TextInput
          label="Nome"
          value={nome}
          onChangeText={(texto) => setNome(texto)}
          style={styles.input}
```

```
        />
        <TextInput
          label="Endereço"
          value={endereco}
          onChangeText={(texto) => setEndereco(texto)}
          style={styles.input}
        />
        <TextInput
          label="Cidade"
          value={cidade}
          onChangeText={(texto) => setCidade(texto)}
          style={styles.input}
        />
        <TextInput
          label="Estado"
          value={estado}
          onChangeText={(texto) => setEstado(texto)}
          style={styles.input}
        />
        <TextInput
          label="CEP"
          value={cep}
          onChangeText={(texto) => setCep(texto)}
          style={styles.input}
        />
        <TextInput
          label="Telefone"
          value={telefone}
          onChangeText={(texto) => setTelefone(texto)}
          style={styles.input}
        />
        <TextInput
          label="E-mail"
          value={email}
          onChangeText={(texto) => setEmail(texto)}
          style={styles.input}
        />
      </View>
      <View style={styles.resumo}>
        <Text style={styles.tituloResumo}>Resumo do Pedido</Text>
        {pedido.produtos.map((produto, index) => (
          <View key={index} style={styles.produto}>
            <Text>{produto.nome}</Text>
            <Text>Quantidade: {produto.quantidade}</Text>
            <Text>Preço: {produto.preco}</Text>
          </View>
        ))}
        <Text style={styles.total}>
          Total: {pedido.produtos.reduce((total, produto) => total +
produto.preco * produto.quantidade, 0)}
        </Text>
```