

Tutorial CRUD em Android / iOS com React Native

Parte 2

João Ozorio

Tutorial CRUD em Android / iOS com React Native

Parte 2

Como havíamos visto na parte anterior, estruturamos a navegação em abas e criamos apenas a aparência do formulário de cadastro.

Nós prosseguiremos. Observe os slides subsequentes.

#2 Formulário

Agora vamos programar os estados dos campos.
Pra quê? Para os valores serem armazenados na
memória quando o usuário digitá-los.

Usando a função useState do React, no AppForm.js:

```
1 | import React, {useState} from 'react';
```

#2 Formulário

Depois, vamos criar nossos objetos de estado e as funções de manipulação do estado, dentro da function AppForm e antes do seu retorno.

```
1 export default function AppForm({ navigation }) {  
2   const [descricao, setDescricao] = useState('');  
3   const [quantidade, setQuantidade] = useState('');  
4  
5   function handleDescriptionChange(descricao){ setDescricao(descricao); }  
6   function handleQuantityChange(quantidade){ setQuantidade(quantidade); }  
7   function handleButtonPress(){  
8     console.log({id: new Date().getTime(), descricao, quantidade});  
9     navigation.navigate("AppList");  
10  }  
11}
```

Essa função também joga o usuário de volta para a aba de listagem, o que será muito útil mais tarde.

Note que ele faz isso usando um objeto navigation, que estou recebendo por parâmetro na function default.

#2 Formulário

Agora, vamos referenciar esta funções de manipulação de estado nas propriedades `onChangeText` dos inputs e `onPress` do botão, como abaixo.

```
1 <TextInput  
2   style={styles.input}  
3   onChangeText={handleDescriptionChange}  
4   placeholder="O que está faltando em casa?"  
5   clearButtonMode="always" />  
6 <TextInput  
7   style={styles.input}  
8   onChangeText={handleQuantityChange}  
9   placeholder="Digite a quantidade"  
10  keyboardType={'numeric'}  
11  clearButtonMode="always" />  
12 <TouchableOpacity style={styles.button} onPress={handleButtonPress}>  
13   <Text style={styles.buttonText}>Salvar</Text>  
14 </TouchableOpacity>
```

Agora, ao atualizar a nossa aplicação React Native, preencher os campos do formulário e clicar no botão de Salvar, o console deve exibir o objeto JSON dos campos mapeados e nos jogar de volta para a aba de listagem.

#2 Formulário - Salvando no Banco de Dados

Agora que temos todo o formulário pronto e funcionando, falta só pegarmos os dados capturados pelos estados e salvar no banco de dados local do dispositivo.

```
1 expo install @react-native-async-storage/async-storage
```

Este AsyncStorage é uma forma bem simples para armazenar dados locais baseado em chave-valor, onde o valor é sempre textual. Sendo sempre textual, o mais comum é a gente salvar objetos JSON em forma de texto, o que chamamos geralmente de objeto “serializado”.

#2 Formulário - Salvando no Banco de Dados

Para usá-lo, vamos alterar os nossos imports do AppForm.js.

```
1 import React, {useState} from 'react';
2 import { StatusBar } from 'expo-status-bar';
3 import { StyleSheet, Text, View, TextInput, TouchableOpacity } from 'react-native';
4 import AsyncStorage from '@react-native-async-storage/async-storage';
```

#2 Formulário - Salvando no Banco de Dados

E agora, na nossa função handleButtonPress que hoje só imprime o objeto no console, vamos usá-la para salvar os dados obtidos no nosso banco de dados.

```
1 | async function handleButtonPress(){
2 |   const listItem = {id: new Date().getTime(), descricao, quantidade: parseInt(quantidade)};
3 |   let savedItems = [];
4 |   const response = await AsyncStorage.getItem('items');
5 |
6 |   if(response) savedItems = JSON.parse(response);
7 |   savedItems.push(listItem);
8 |
9 |   await AsyncStorage.setItem('items', JSON.stringify(savedItems));
10|   navigation.navigate("AppList", listItem);
11| }
```

Agora você pode testar, cadastrando uma série de itens, mas somente vai ver alguma coisa acontecendo no console, pois a tela de listagem ainda não funciona, o que faremos na sequência.

#2 Criando a Listagem

Uma vez que já temos o cadastro de itens da nossa lista de compras funcionando, podemos construir a outra tela, de listagem.

Criar ela será um pouco mais complexo que a de formulário, pois ela é uma listagem de itens e, quando temos uma listagem de itens, temos de criar um componente para o item primeiro.

Vamos criar um novo arquivo chamado “AppItem.js”.

```
1 import React from 'react';
2 import {StyleSheet, Text, View, TouchableOpacity} from 'react-native';
3
4 export default function AppItem(props){
5   return (
6     <View style={styles.container}>
7       <Text style={styles.textItem}>{props.item}</Text>
8       <View style={styles.buttonsContainer}>
9         <TouchableOpacity style={styles.deleteButton} >
10           <Text style={styles.buttonText}>X</Text>
11         </TouchableOpacity>
12         <TouchableOpacity style={styles.editButton} >
13           <Text style={styles.buttonText}>Editar</Text>
14         </TouchableOpacity>
15       </View>
16     </View>
17   );
18 }
```

Os imports são auto-explicativos, mas a function default requer atenção. Primeiro, ela irá receber um objeto props que conterá o texto do item a ser adicionado na listagem.

Além do texto, temos dois botões, um de editar o item e outro de excluir.

#2 Criando a Listagem

Abaixo, o estilo deste mesmo componente AppItem.js.

```
1 const styles = StyleSheet.create({
2   container: {
3     backgroundColor: '#fff',
4     marginTop: 20,
5     width: '100%'
6   },
7   buttonsContainer: {
8     flexDirection: 'row-reverse',
9     alignItems: 'flex-end',
10    borderBottomWidth: 1,
11    borderBottomColor: '#CCC',
12    paddingBottom: 10,
13    marginTop: 10,
14  },
15  editButton: {
16    marginLeft: 10,
17    height: 40,
18    backgroundColor: 'blue',
19    borderRadius: 10,
20    padding: 10,
21    fontSize: 12,
22    elevation: 10,
23    shadowOpacity: 10,
24    shadowColor: '#ccc',
25    alignItems: 'center'
26  },
27  deleteButton: {
28    marginLeft: 10,
29    height: 40,
30    width: 40,
31    backgroundColor: 'red',
32    borderRadius: 10,
33    padding: 10,
34    fontSize: 12,
35    elevation: 10,
36    shadowOpacity: 10,
37    shadowColor: '#ccc',
38    alignItems: 'center'
39  },
40  buttonText: {
41    color: '#fff',
42    fontWeight: 'bold',
43  },
44  textItem: {
45    fontSize: 20,
46  }
47});
```

#2 Criando a Listagem

Agora que temos o nosso item da lista, vamos construir a lista em si, representada no módulo AppList.js.

Primeiro, vamos revisar os imports deste arquivo, adicionando alguns itens novos que usaremos a seguir.

```
1 import { StatusBar } from 'expo-status-bar';
2 import React, {useState} from 'react';
3 import { StyleSheet, View, Text, ScrollView } from 'react-native';
4 import AppItem from './AppItem';
```

#2 Criando a Listagem

Segundo, vamos adicionar na função default desta tela um estado para o array de itens que, em um primeiro momento, estarão estáticos, mas que futuramente virão do banco de dados.

```
1 const [items, setItems] = useState([
2   {id: 1, quantidade: 5, descricao: "arroz" },
3   {id: 2, quantidade: 1, descricao: "feijão" },
4   {id: 3, quantidade: 0.5, descricao: "lentilha" },
5   {id: 4, quantidade: 1, descricao: "massa" },
6   {id: 5, quantidade: 1, descricao: "ketchup" },
7   {id: 6, descricao: "queijo-rolado" }
8 ]);
```

#2 Criando a Listagem

Com esse estado pronto, vamos mudar a interface desta tela para que tenha uma ScrollView (área com rolagem) e dentro dela, vamos executar um código JavaScript que vai adicionar itens dinamicamente conforme o conteúdo de um array que definimos um pouco antes do return com conteúdos estáticos, apenas para podermos visualizar antes de tornar esta página realmente dinâmica.

```
1 | return (
2 | <View style={styles.container}>
3 |   <StatusBar style="light" />
4 |   <Text style={styles.title}>Lista de Compras</Text>
5 |   <ScrollView
6 |     style={styles.scrollContainer}
7 |     contentContainerStyle={styles.itemsContainer}>
8 |     { items.map(item => {
9 |       return <AppItem key={item.id} id={item.id} item={`${item.quantidade} de ${item.de
10 |     }) }
11 |   </ScrollView>
12 | </View>
13 | );
```

Note também como usei o nosso componente AppItem, passando propriedades pra ele, oriundos dos itens do array armazenado no estado que criamos antes. A propriedade key é obrigatória e usaremos ela mais tarde, na listagem e exclusão.

#2 Criando a Listagem

Você vai precisar de alguns estilos novos também, que pode conferir abaixo.

```
1 const styles = StyleSheet.create({
2   container: {
3     flex: 1,
4     backgroundColor: '#D93600',
5     alignItems: 'center',
6     justifyContent: 'center'
7   },
8   title: {
9     color: '#fff',
10    fontSize: 20,
11    fontWeight: 'bold',
12    marginTop: 50,
13    marginBottom: 20
14  },
15  scrollContainer: {
16    flex: 1,
17    width: '90%'
18  },
19  itemsContainer: {
20    flex: 1,
21    marginTop: 10,
22    padding: 20,
23    borderTopLeftRadius: 10,
24    borderTopRightRadius: 10,
25    alignItems: 'stretch',
26    backgroundColor: '#fff'
27  },
28});
```

Aqui não tem absolutamente nenhuma novidade em relação ao que já fizemos antes.

#2 Criando a Listagem

Como resultado, você deve ter uma tela de listagem com a seguinte aparência.



O fim

E com isso, encerramos a segunda parte do nosso tutorial.

Hoje fizemos um pouco do código e os estilos, nada muito difícil, ou que passasse da curva de aprendizado.

Você está quase lá, pupilo. Não desista.

Sua motivação não é a conclusão. A motivação é algo que você colhe ao longo do tempo, junto com todo o conteúdo aprendido nessas duas partes. Novidades virão em breve.

Que a força esteja com você.