1. Atividades Práticas - React Hooks

Nesta atividade prática, vamos trabalhar com os conceitos de Rect Hooks. Nesta atividade prática, vamos desenvolver uma aplicação de Todo-List.

1.1 Criando nosso Projeto

Vamos criar um projeto chamado my-todo, digitando este comando no terminal:

```
expo init my-todo
```

Em seguida, algumas opções serão apresentadas. Selecione a opção “blank”. O Expo será executado no diretório que você especificar. Este processo leva alguns segundos e, logo após terminar, digite o seguinte comando para entrar no diretório recém criado do nosso projeto:

```
cd my-todo
```

Você agora está dentro da raiz do seu projeto. Até aqui, você criou um projeto e adicionou todas as dependências. Você pode, agora, abrir a aplicação no editor Visual Studio Code. Para isso, dentro do diretório `my-todo` digite o seguinte comando:

```
code .
```

O Visual Studio Code deverá abrir com a pasta raiz do seu projeto sendo acessada. Em seguida, você inicializará um servidor de desenvolvimento local. Assim, execute o seguinte comando:

```
expo start
```

Ao executar esse script, o Expo CLI inicia o Metro Bundler, que é um servidor HTTP que compila o código JavaScript de nosso aplicativo (usando o Babel) e o serve ao aplicativo Expo.

1.2 Criando um estilo para sua aplicação

Agora, vá para o arquivo App.js e adicione os seguintes estilos que usaremos em todo o aplicativo. Cabe lembrar que o estilo não será o foco desta aula. Veja como deve ficar:

```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
 return (
   <View style={styles.container}>
     <Text>Open up App.js to start working on your app!</Text>
     <StatusBar style="auto" />
   </View>
 );
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: "#20FFee",
   padding: '30',
   alignItems: 'center',
   justifyContent: 'center',
 },
 todoList: {
   backgroundColor: '#e8e8e8',
   borderRadius: '4',
   padding: '5',
   maxWidth: '400'
  },
  todo: {
   backgroundColor: '#fff',
   padding: '3',
```

```
    fontSize: '12',
    marginBottom: '6',
    borderRadius: '3',
    alignItems: 'center'
  },
  button: {
    padding: '3',
    fontSize: '12',
    marginBottom: '6',
    borderRadius: '3',
    alignItems: 'center'
  }

});
```

1.3 Lendo uma lista de itens

Com seu aplicativo em execução e o estilo pronto para ser usado, vamos começar na parte ler uma lista de itens do Todo-List. Ou seja, queremos fazer uma lista de coisas para que possamos ler/visualizar a lista.

1.3.1 Adicionando no estado interno

Ainda no arquivo App.js, adicione um estado ao nosso componente. Como usaremos React Hooks, o estado parecerá um pouco diferente do que usamos com classes.

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


export default function App() {
 const [todos, setTodos] = useState([
    { text: "Aprender sobre o React" },
    { text: "Encontrar um amigo para o almoço" },
    { text: "Passar no supermercado" }
```

```
 ]);
 return (
   <View style={styles.container}>
   </View>
 );
...
```

Observe que componente que criamos é um componente funcional. Nas versões anteriores do React, os componentes funcionais não conseguiam lidar com o estado. Mas agora, usando Hooks, eles podem.

Desta forma, em nosso exemplo:

- O primeiro parâmetro, `todos`, usamos para nomear nosso estado.
- O segundo parâmetro, `setTodos`,usamos para definir o estado.

O hook `useState` é o que o React usa para se conectar ao estado do componente. Em seguida, criamos um array de objetos e temos o início de nosso estado. Ainda, observe que estamos com a `<View>` vazia (neste momento).

1.3.3 Criando o componente Todo

Após, queremos criar um componente que possamos usar posteriormente no retorno do componente App. Cabe lembrar que podemos criar diversos componentes em um mesmo arquivo. Desta forma, vamos criar o componente Todo no arquivo App.js. Este componente mostrará a parte "`text`" do valor de `todo` recebido (`todo.text`), assim:

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

const Todo = ({ todo }) => <View
style={styles.todo}>{todo.text}</View>;

export default function App() {
...
```

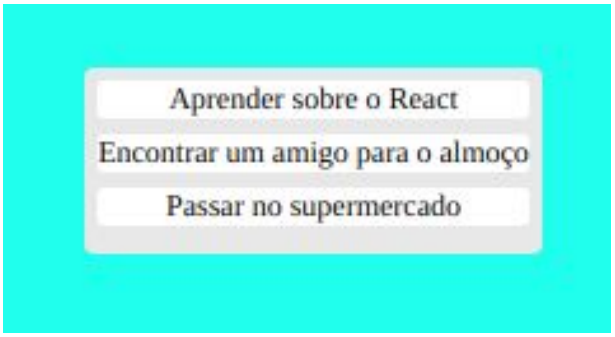
1.3.4 Obtendo a lista de itens de Todo

Em seguida, devemos instanciar o componente Todo no componente `App`. Logo, desça até a parte de retorno do componente `App`, onde temos `<View></View>`. Queremos que a nossa lista seja exibida na página.

```
import React, { useState } from "react";
import { StyleSheet, View } from "react-native";

const Todo = ({ todo }) => <View
style={styles.todo}>{todo.text}</View>;

export default function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React" },
   { text: "Encontrar um amigo para o almoço" },
   { text: "Passar no supermercado" },
 ]);

 return (
   <View style={styles.container}>
     <View style={styles.todoList}>
       {todos.map((todo, index) => (
         <Todo key={index} index={index} todo={todo} />
       ))}
     </View>
   </View>
 );
}
...
```

Assim, usando o método JavaScript chamado `map()`, conseguimos criar um novo array de itens, mapeando os itens do estado interno na variável `todo` e exibindo-os de acordo com seu índice.

Veja no navegador o resultado até o momento:



1.3.5 Criando novos itens na lista de tarefas

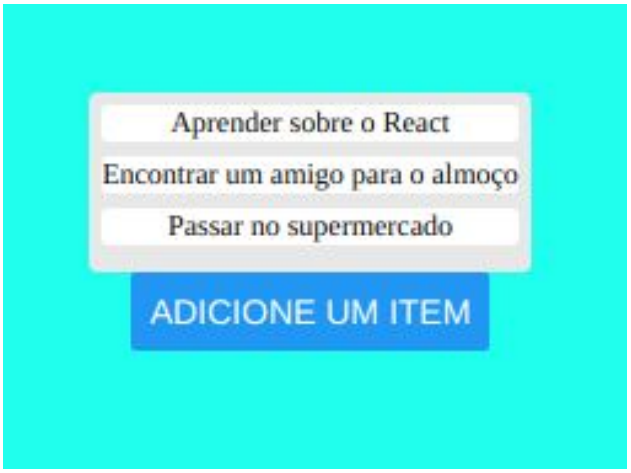
Vamos codificar a parte da aplicação que permite criar um novo item na lista de tarefas. Assim, vamos criar a função `addTodo` dentro do componente `App`. Permanecendo no App.js, essa a função deverá pegar a lista de itens existente, adicionar o novo item e exibir essa nova lista. Vamos criar um botão para adicionar um item qualquer, neste caso “Mais um item”.

```
...
export default function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React" },
   { text: "Encontrar um amigo para o almoço" },
   { text: "Passar no supermercado" },
 ]);

 const addTodo = info => {
   const newTodos = [...todos, { text: info }];
   setTodos(newTodos);
 };



 return (
```

```
    <View style={styles.container}>
      <View style={styles.todoList}>
        {todos.map((todo, index) => (
          <Todo key={index} index={index} todo={todo} />
        ))}
      </View>
      <Button style={styles.button}
        onPress={() => addTodo("Mais um item")}
        title='Adicione um item'
      />
    </View>
  );
}
...
```

Observe que usamos o spread operator. Os três pontos antes de `todos` copia os itens da lista, para que possamos adicionar o novo item de tarefa. Em seguida, atualizaremos o estado com `setTodos`.

Agora você pode adicionar um item de tarefa à sua lista no seu navegador.



1.3.6 Atualizando itens na lista de tarefas

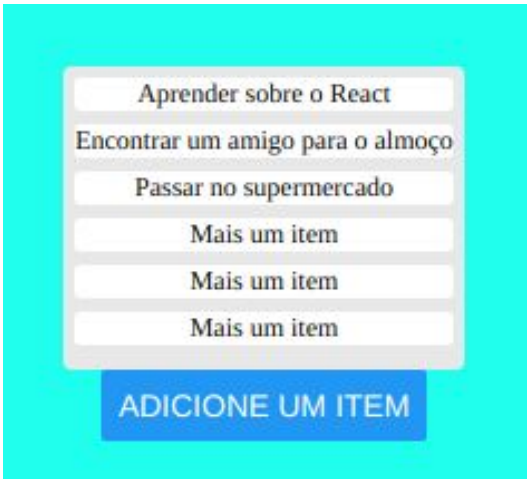
Vamos, agora, adicionar a funcionalidade para “riscar” um item em nossa lista de tarefas. Para realizar isso, o estado em nosso componente `App` precisa de uma informação sobre o

status de cada item, uma vez que precisamos informar que o status esteja "Concluído". Logo, ao adicionar a informação `isCompleted` (que inicia for igual a `false)` ao array.

```
...
export default function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
     const newTodos = [...todos, { text: info, isCompleted:
false }];
   setTodos(newTodos);
 };
...
```

Observe que também precisamos alterar a função `addTodo` para adicionar esta informação sobre o `status`.

Veja o resultado na tela ao clicar no botão ADICIONE UM ITEM:



Agora, dentro do componente App, vamos precisar de uma nova função, chamada de `completeTodo`, que seja capaz de "concluir" um item. Vamos adicionar esta função no componente funcional `Todo`. Para isto, usar o operador spread para pegar a lista atual de itens. Nesta função, alteraremos o status `isCompleted` para `true`, para que ele saiba que um item foi concluído. Ele atualizará o estado e o definirá para o `newTodos`.

```
...
const completeTodo = index => {
   const newTodos = [...todos];
   newTodos[index].isCompleted = true;
   setTodos(newTodos);
 };
...
```

Veja abaixo que, quando o botão Concluir é clicado, ele adiciona o estilo de decoração de texto e riscará o item. Estamos usando um operador ternário, um recurso no JavaScript ES6, que é outra maneira de fazer uma declaração `if/else`. Assim, podemos “concluir” um item na lista e "atualizar" esta lista.

```
import React, { useState } from "react";
import { StyleSheet, Button, View, Text } from "react-native";


function Todo({ todo, index, completeTodo }) {
 return (
  <View style={styles.todo}>
          <Text  style={{  textDecoration:  todo.isCompleted  ?
"line-through" : "" }}>
     {todo.text}
    </Text>
    <Button style={styles.button}
       onPress={() => completeTodo(index)}
       title='Concluir'
     />
```

```
  </View>
 );
}
export default function App() {
...
```

Em seguida, desça até o retorno do componente `App` e adicione a informação que passe `completeTodo` para o componente `Todo`:

```
...
export default function App() {
 ...
 return (
   <View style={styles.container}>
     <View style={styles.todoList}>
       {todos.map((todo, index) => (
         <Todo key={index}
         index={index}
         todo={todo}
         completeTodo={completeTodo}
       />
       ))}
    ...
```

Ou seja, até o momento, nosso código deve estar assim:

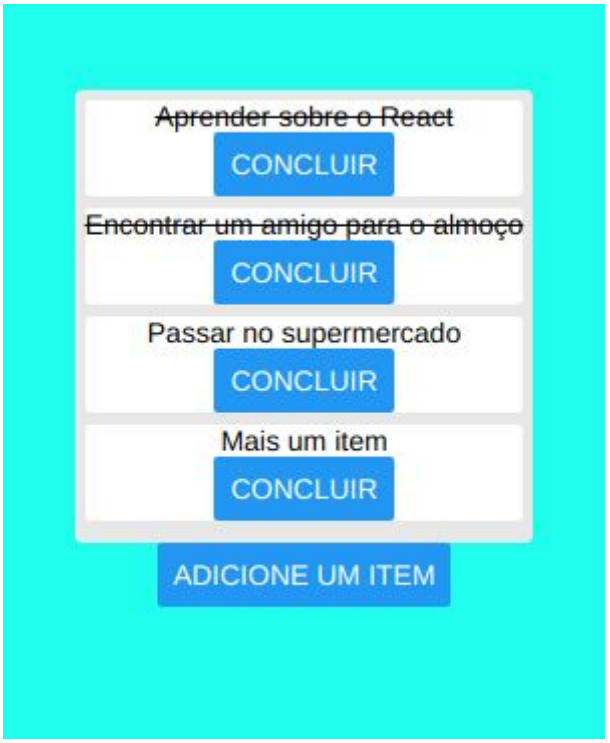
```
import React, { useState } from "react";
import { StyleSheet, Button, View, Text } from "react-native";

function Todo({ todo, index, completeTodo }) {
 return (
  <View style={styles.todo}>
```

```
        <Text  style={{  textDecoration:  todo.isCompleted  ?
"line-through" : "" }}>
    {todo.text}
   </Text>
   <Button style={styles.button}
      onPress={() => completeTodo(index)}
      title='Concluir'
    />
  </View>
 );
}
export default function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
    const  newTodos  =  [...todos,  {  text:  info,  isCompleted:
false }];
   setTodos(newTodos);
 };
 const completeTodo = index => {
   const newTodos = [...todos];
   newTodos[index].isCompleted = true;
   setTodos(newTodos);
 };
```

```
 return (
   <View style={styles.container}>
     <View style={styles.todoList}>
       {todos.map((todo, index) => (
         <Todo key={index}
         index={index}
         todo={todo}
         completeTodo={completeTodo}
       />
       ))}
     </View>
     <Button style={styles.button}
       onPress={() => addTodo("Mais um item")}
       title='Adicione um item'
     />
   </View>
 );
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: "#20FFee",
   padding: "30",
   alignItems: "center",
   justifyContent: "center",
 },
```

```
  todoList: {
    backgroundColor: "#e8e8e8",
    borderRadius: "4",
    padding: "5",
    maxWidth: "400",
  },
  todo: {
    backgroundColor: "#fff",
    padding: "3 10",
    fontSize: "12",
    marginBottom: "6",
    borderRadius: "3",
    alignItems: "center",
  },
  button: {
    padding: "3 10",
    fontSize: "12",
    marginBottom: "6",
    borderRadius: "3",
    alignItems: "center",
  },
});
```

Veja que sua lista de tarefas deve estar funcionando!



Agora podemos ler nossa lista, adicionar um item à nossa lista e atualizar o status completo de cada item. Em seguida, adicionaremos a funcionalidade de exclusão.

1.4 Excluindo um item da lista de tarefas

Dentro do componente `App`, vamos criar a função `removeTodo` para que, quando clicarmos em um "Remover" para excluir um item, ele seja excluído. Nesta função `removeTodo`, usaremos novamente o operador spread. Mas, assim que pegarmos a lista atual, separaremos o índice escolhido do array de itens. Uma vez removido, retornaremos o novo estado, configurando-o para ser `newTodos`, através de `setTodos`.

```
...
 const removeTodo = index => {
   const newTodos = [...todos];
   newTodos.splice(index, 1);
   setTodos(newTodos);
 };
...
```

Em nosso componente Todo, você precisamos adicionar um botão que chame `removeTodo`.

```
...
function Todo({ todo, index, completeTodo,removeTodo }) {
 return (
  <View style={styles.todo}>
         <Text  style={{  textDecoration:  todo.isCompleted  ?
"line-through" : "" }}>
     {todo.text}
    </Text>
    <Button style={styles.button}
       onPress={() => completeTodo(index)}
       title='Concluir'
     />
     <Button style={styles.button}
       onPress={() => removeTodo(index)}
       title='Remover'
     />
...
```

Em seguida, desça até o retorno do componente `App` e adicione a informação que passe `removeTodo` para o componente `Todo`:

```
...
return (
   <View style={styles.container}>
     <View style={styles.todoList}>
       {todos.map((todo, index) => (
         <Todo key={index}
         index={index}
         todo={todo}
         completeTodo={completeTodo}
         removeTodo={removeTodo}
```

```
        />
        ))}
      </View>
...
```

Com isso adicionado, acesse seu navegador e você verá um botão com um "Remover" que, quando clicado, exclui completamente o item.

O nosso código final é o seguinte:

```
import React, { useState } from "react";
import { StyleSheet, Button, View, Text } from "react-native";


function Todo({ todo, index, completeTodo,removeTodo }) {
 return (
  <View style={styles.todo}>
         <Text  style={{  textDecoration:  todo.isCompleted  ?
"line-through" : "" }}>
     {todo.text}
    </Text>
```

```
    <Button style={styles.button}
      onPress={() => completeTodo(index)}
      title='Concluir'
    />
    <Button style={styles.button}
      onPress={() => removeTodo(index)}
      title='Remover'
    />
  </View>
 );
}
export default function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
    const newTodos = [...todos, { text: info, isCompleted:
false }];
   setTodos(newTodos);
 };
 const completeTodo = index => {
   const newTodos = [...todos];
   newTodos[index].isCompleted = true;
   setTodos(newTodos);
 };
```

```
 const removeTodo = index => {
   const newTodos = [...todos];
   newTodos.splice(index, 1);
   setTodos(newTodos);
 };



 return (
   <View style={styles.container}>
     <View style={styles.todoList}>
       {todos.map((todo, index) => (
         <Todo key={index}
         index={index}
         todo={todo}
         completeTodo={completeTodo}
         removeTodo={removeTodo}
       />
       ))}
     </View>
     <Button style={styles.button}
       onPress={() => addTodo("Mais um item")}
       title='Adicione um item'
     />
   </View>
 );
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
```

```
    backgroundColor: "#20FFee",
    padding: "30",
    alignItems: "center",
    justifyContent: "center",
  },
  todoList: {
    backgroundColor: "#e8e8e8",
    borderRadius: "4",
    padding: "5",
    maxWidth: "400",
  },
  todo: {
    backgroundColor: "#fff",
    padding: "3",
    fontSize: "12",
    marginBottom: "6",
    borderRadius: "3",
    alignItems: "center",
  },
  button: {
    padding: "3",
    fontSize: "12",
    marginBottom: "6",
    borderRadius: "3",
    alignItems: "center",
  },
});
```

Desta forma, nesta atividade prática você criou uma aplicação de lista de tarefas com React Hooks