Desenvolvimento para Dispositivos Móveis

Aula 07 - Listas

Material Didático do Instituto Metrópole Digital - IMD

Apresentação

Nesta aula, continuaremos estudando alguns componentes *react native*.

Objetivos

- Conhecer como organizar uma lista de objetos em tela
- Como utilizar a listagem em dispositivos móveis
- Conhecer como dividir uma lista em seções

ScrollView

Link do video da aula: https://youtu.be/smMGtfngceM

Continuando o estudo nos componentes, veremos nesta aula, em particular, o *ScrollView*, que é o componente de rolagem de tela.

Criando parágrafos

Com o arquivo "App.js" reduzido apenas com o básico, um parágrafo aleatório, coloque-o para renderizar na tela e aumente a fonte. Ao fazer isso, teremos um longo texto sendo impresso na tela do dispositivo.

Exemplo de parágrafo abaixo:

> "At that moment he had a thought that he'd never imagine he'd consider. "I could just cheat," he thought, "and that would solve the problem." He tried to move on from the thought but it was persistent. It didn't want to go away and, if he was honest with himself, he didn't want it to. "

Criando barra de rolagem

Em dispositivos móveis temos de usar um componente para que seja possível rolar objetos em tela, sejam eles textos, imagens ou qualquer mídia. Para fazer isso, utilizamos o componente '***ScrollView***', que engloba entre a abertura e o fechamento todos os objetos que devem permitir o *scroll* na tela.

Acompanhando a aula, implemente o código abaixo:

```
import React from 'react';
import { SafeAreaView, ScrollView, Text } from 'react-native';
```

```
export default function App() {
  return (
    <SafeAreaView>
      <ScrollView style={{ margin: 16 }}>
        <Text style={{ fontSize: 28 }}>At that moment he had a
thought that he'd never imagine he'd consider. "I could just cheat,"
he thought, "and that would solve the problem." He tried to move on
from the thought but it was persistent. It didn't want to go away
and, if he was honest with himself, he didn't want it to.</Text>
        <Text style={{ fontSize: 28 }}>At that moment he had a
thought that he'd never imagine he'd consider. "I could just cheat,"
he thought, "and that would solve the problem." He tried to move on
from the thought but it was persistent. It didn't want to go away
and, if he was honest with himself, he didn't want it to.</Text>
        <Text style={{ fontSize: 28 }}>At that moment he had a
thought that he'd never imagine he'd consider. "I could just cheat,"
he thought, "and that would solve the problem." He tried to move on
from the thought but it was persistent. It didn't want to go away
and, if he was honest with himself, he didn't want it to.</Text>
      </ScrollView>
    </SafeAreaView>
  );
}
```

FlatList

Link do video da aula: https://youtu.be/yb-V_EWInzA

Nesta videoaula, aula continuaremos estudando os componentes do *react native*, em particular, o ***FlatList*** .

FlatList

É muito comum precisarmos listar algo em dispositivos móveis e é importante que ao fazer isso, seja utilizado o componente adequado. Além disso, em dispositivos móveis a memória é algo que precisamos ter bastante cuidado, isso porque os dispositivos móveis têm pouca memória disponível.

Então, para resolver isso, os dispositivos têm um cuidado refinado com esse detalhe, mantendo em memória apenas o que está sendo exibido na tela. Dito isso, utilizaremos o *FlatList* para esse propósito.

Simulando os dados

Para utilizar o *FlatList* precisamos simular uma lista de dados que será passada como parâmetro para o componente em questão. Então, criaremos o *array* abaixo:

```
const DATA = [
  { id: 0, title: "Primeiro item" },
  { id: 1, title: "Segundo item" },
  { id: 2, title: "Terceiro item" }
]
```

Criando função de renderização

Outro parâmetro importante é o *renderItem* que recebe um componente que será utilizado para representar a renderização de um único item, uma espécie de modelo.

Então, criaremos o componente abaixo:

```
function Item(props) {
  return (
    <View style={{ height: 30 }}>
      <Text>{props.title}</Text>
    </View>
  )
}
```

Identificador único

Outro parâmetro importante é o *KeyExtractor* , que é uma forma de identificar unicamente cada elemento e saber quais elementos renderizar, para melhor gerenciamento de memória.

Esse identificador deve ser único para cada elemento, não podendo haver dois itens com a mesma *KeyExtrator*.

Melhorando a visualização

Após configurado o *FlatList* será utilizada uma pequena estilização para melhorar a visualização dos dados.

Acompanhando a aula, ao fina seu código deverá ficar como se vê abaixo:

```
import React from 'react';
import { FlatList, SafeAreaView, View, Text } from 'react-native';
```

```
const DATA = [
  { id: 0, title: "Primeiro item" },
  { id: 1, title: "Segundo item" },
  { id: 2, title: "Terceiro item" },
  { id: 3, title: "Quarto item" }
]

function Item(props) {
  return (
    <View style={{ height: 30, backgroundColor: '#AFC', padding: 10,
margin: 10 }}>
      <Text>{props.title}</Text>
    </View>
  )
}

export default function App() {
  return (
    <SafeAreaView>
      <FlatList
        data={DATA}
        renderItem={it => <Item title={it.item.title} />}
        keyExtractor={it => it.id}
      >

      </FlatList>
    </SafeAreaView>
  );
}
```

Nota:

Em alguns sistemas, ao executar o aplicativo, pode ser lançada a seguinte exceção: ***Warning: Failed child context type: Invalid child context `virtualizedCell.cellKey` of type `number` supplied to `CellRenderer`, expected `string`.***

Nesses casos, o conjunto de dados usados para simular a lista deve conter aspas no 'id'. Veja o exemplo abaixo:

```
const DATA = [
  { id: "0", title: "Primeiro item" },
  { id: "1", title: "Segundo item" },
  { id: "2", title: "Terceiro item" },
```

```
  { id: "3", title: "Quarto item" }
]
```

Section List

Link do video da aula: https://youtu.be/GDCmcfs5zTI

Continuando o conhecimento sobre listas, entraremos agora no * Section List**.

Section List

O que difere o *FlatList* do *Section List* é o fato de que um faz a listagem em sequência e o outro divide em seções em sua visualização.

Simulando dados

Assim como na listagem com *FlatList*, precisamos de dados para listar na tela. Para fazer isso, iremos criar um conjuto de dados, porém, dessa vez terá um novo formato: cada seção terá um título e um conjunto de dados associados a ela.

Abaixo podemos ver um exemplo:

```
const DATA = [
  { title: "Pratos principais", data: ["Pizza", "Burger", "Risoto"]
},
  { title: "Entradas", data: ["Batata Frita", "Onion Rings"] },
  { title: "Sobremesas", data: ["Sorvete de Creme", "Petit Gateau"]
}
]
```

Sections

Assim como no *FlatList*, o *SectionList* necessita de uma pequena configuração que é feita através dos parâmetros.

RenderItem

Assim como no *FlatList*, existe o parâmetro *renderItem*, o qual recebe um componente que será utilizado para representar a renderização de um único item, uma espécie de modelo.

Então, criaremos o componente abaixo:

```
function Item(props) {
  return (
    <View style={{ height: 30, backgroundColor: '#AFC', padding: 10,
margin: 10 }}>
      <Text>{props.title}</Text>
    </View>
  )
}
```

RenderSectionHeader

Até agora, não dividimos nossa lista em seções no momento da visualização. Para fazer isso, precisamos desse terceiro parâmetro que recebe também um componente.

KeyExtractor

É usado pela mesma razão dita no *FlatList*: identificação única de cada objeto.

Utilizando *Section List*

Acompanhando toda a aula, configurando o componente e implementando o código, ao final, seu arquivo deverá ser como se observa abaixo:

```
import React from 'react';
import { SafeAreaView, View, Text, SectionList } from 'react-
native';

const DATA = [
  {
    title: "Pratos principais",
    data: ["Pizza", "Burger", "Risoto"]
  },
  {
    title: "Entradas",
    data: ["Batata Frita", "Onion Rings"]
  },
  {
    title: "Sobremesas",
    data: ["Sorvete de Creme", "Petit Gateau"]
  }
]

function Item(props) {
```

```
  return (
    <View style={{ height: 60, backgroundColor: '#AFC', padding: 10,
margin: 10 }}>
      <Text>{props.title}</Text>
    </View>
  )
}

export default function App() {
  return (
    <SafeAreaView>
      <SectionList
        style={{ margin: 8 }}
        sections={DATA}
        renderItem={it => <Item title={it.item} />}
        renderSectionHeader={(item) => <Text style={{ fontSize: 20
}}>{item.section.title}</Text>}
        keyExtractor={(item, index) => item + index}
      />
    </SafeAreaView>
  );
}
```

Resumo

Nesta aula, foi visto como trabalhar com a organização de objetos em listas, seja de maneira sequencial ou divididos em seções. Ademais, os métodos abordados têm suporte para trabalhar em dispositivos móveis, visto que é preciso maior atenção por questões de memória.