

# 1. Atividades Práticas - Estilização de Componentes

---

Nesta atividade prática, vamos trabalhar com os conceitos de estilização de componentes do React Native. Nesta atividade prática, vamos desenvolver uma aplicação de bloco de anotações.

## 1.1 Introdução

No React Native, podemos usar StyleSheets para escrever nosso estilo. StyleSheets pode ser considerado um subconjunto de CSS junto com alguns helpers adicionais para React Native. StyleSheets, ao contrário do CSS, são objetos JavaScript puros. Assim, StyleSheets são usados no React Native de forma semelhante aos estilos inline na web.

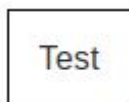
Por exemplo, para adicionar preenchimento (`padding`) e borda (`border`) em CSS a uma tag `<span>`, escreveremos uma classe assim:

```
.button {  
  padding: 10px;  
  text-align: center;  
  border: 1px solid black;  
}
```

Em seguida, adicionaremos essa classe ao nosso `<span>` desta forma:

```
<span class="button"> Test </span>
```

E o resultado na tela seria:



No React Native, não há conceito de pixels ou classes. Em vez disso, nossos tamanhos serão especificados em "unidades" que serão convertidas automaticamente em pixels, com base na densidade de pixels da tela, pelo React Native.

Se escrevermos a mesma View com StyleSheet do React Native, teremos:

```
import { StyleSheet } from 'react-native';
export default StyleSheet.create({
  button: {
    padding: 10,
    textAlign: 'center',
    borderWidth: 1,
    borderColor: 'black'
  }
});
```

Esses estilos podem ser adicionados à nossa View usando:

```
import styles from './styles.js';
...
<View style={styles.button}></View>
```

Observe que, como estamos escrevendo objetos JavaScript:

- Escrevemos atributo `style` usa uma versão camel-cased compatível com a do CSS
- Importamos nosso StyleSheet em nossos componentes e os usamos declarando um atributo `style` em vez de classes.
- Não há atalhos que usamos no CSS, tal como: `border: '1px solid black'`

## 1.2 Variáveis de Tema da Aplicação

Geralmente, cada aplicativo deve especificar tamanhos de fonte, cores, espaçamentos, etc.. Isso é feito para que o aplicativo pareça consistente em todas as telas. Agora, isso pode ser alcançado mantendo uma convenção em todo o aplicativo. Por exemplo, os designers do projeto podem decidir `fontSize` da seguinte forma para o aplicativo que estamos desenvolvendo:

- 16 - Grande
- 14 - Médio
- 12 - Pequeno

Neste caso, a nossa folha de estilo (um arquivo `style.js`) pode ter a seguinte aparência:

```
import { StyleSheet } from 'react-native';
export default StyleSheet.create({
  largeButtonText: {
    fontSize: 16,
    fontWeight: 'bold'
  },
  largeHeaderText:{
    fontSize: 16
  },
  mediumHeaderText: {
    fontSize: 14,
    color:'blue'
  }
});
```

Embora essa convenção seja boa e ajude você a manter a consistência em tamanhos de texto pequenos, médios e grandes no aplicativo, ela tem algumas falhas fundamentais:

- A equipe de marketing pode, por exemplo, apresentar uma mudança no requisito de que o tamanho da fonte grande deve ser 18 em vez de 16. Agora, como desenvolvedor, você precisará fazer alterações em todo o aplicativo e substituir todas as instâncias de `fontSize: 16` por `fontSize: 18`, o que é um problema.
- Ou, um novo desenvolvedor que se junta à equipe pode não estar ciente de todas as convenções seguidas pela equipe e pode criar um componente com `fontSize` diferente de 12, 14 ou 16.

Para resolver os problemas mencionados acima, podemos criar um arquivo de tema (`theme.style.js`, por exemplo) para a nossa aplicação. Neste arquivo de tema, definimos nossas variáveis de tema da seguinte maneira:

```
export default {
  FONT_SIZE_SMALL: 12,
  FONT_SIZE_MEDIUM: 14,
  FONT_SIZE_LARGE: 16,
  PRIMARY_COLOR: 'rgb(30, 147, 242)',
  SECONDARY_COLOR: 'rgb(238, 167, 2)'
```

```
FONT_WEIGHT_LIGHT: 200,  
FONT_WEIGHT_MEDIUM: 600,  
FONT_WEIGHT_HEAVY: 800  
};
```

Assim, podemos alterar o nosso arquivo de estilo (o style.js) da seguinte forma:

```
import { StyleSheet } from 'react-native';  
import theme from './theme.style.js';  
export default StyleSheet.create({  
  largeButtonText: {  
    fontSize: theme.FONT_SIZE_LARGE,  
    fontWeight: theme.FONT_WEIGHT_HEAVY  
  },  
  largeHeaderText: {  
    fontSize: theme.FONT_SIZE_LARGE  
  },  
  mediumHeaderText: {  
    fontSize: theme.FONT_SIZE_MEDIUM,  
    color: theme.PRIMARY_COLOR  
  }  
});
```

Agora, nosso arquivo de tema determina o tamanho das fontes e a cor primária, etc.

Isso nos dá dois benefícios:

- Se nossa equipe de marketing, por exemplo, nos diz agora para alterar os tamanhos de fonte, podemos alterar as variáveis do tema em um lugar e isso se reflete em todo o aplicativo.
- Isso nos permitirá escrever vários arquivos de tema. Por exemplo, podemos escrever dois arquivos de tema - um para um tema claro e outro para um tema escuro e dar aos usuários do nosso aplicativo a opção de alternar entre os temas.

Bem, agora vamos criar o nosso projeto de exemplo.

## 1.3 Criando nosso Projeto

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

```
expo init my-style
```

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-style
```

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-style` 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.

## 1.4 Criando nosso componente

Inicialmente, vamos criar um componente para a nossa aplicação. Assim, crie um diretório chamado components. Dentro dele, crie um diretório chamado Home e depois um arquivo chamado index.js.

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

function Home() {
  const [state, setState] = useState({text: ''});
  return (
```

```

<View>
  <Text> Digite sua anotação aqui:</Text>
  <TextInput
    multiline={true}
    numberOfLines={4}
    onChangeText={ (text) => setState({text}) }
    value={state.text}/>
</View>
);
}
export default Home;

```

Observe que criamos um componente simples com um texto e um campo para entrada de dados.

Agora, vamos chamar este componente em App.js

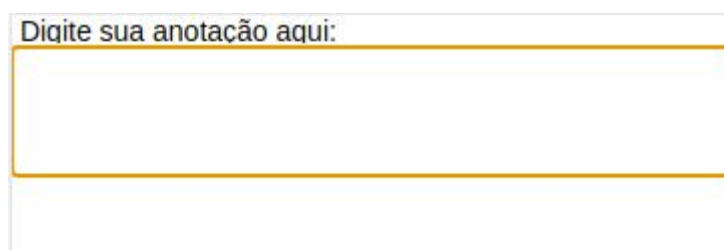
```

import React from "react";
import Home from "../components/home";

export default function App() {
  return (
    <Home/>
  );
}

```

Veja o resultado na tela:


 A screenshot of a web application interface. It features a light gray rectangular container. Inside the container, at the top, is the text "Digite sua anotação aqui:" in a dark gray font. Below this text is a large, empty text input field with a thin orange border. The input field is currently empty, showing only the placeholder text.

## 1.5 Adicionando Estilos ao Projeto

Primeiro de tudo, crie um diretório `style` na raiz do projeto e, após, adicione o arquivo `theme.style.js`.

```
export default {  
  PRIMARY_COLOR: '#2aabb8',  
  FONT_SIZE_SMALL: 12,  
  FONT_SIZE_MEDIUM: 14,  
  FONT_SIZE_LARGE: 16,  
  FONT_WEIGHT_LIGHT: '200',  
  FONT_WEIGHT_MEDIUM: '500',  
  FONT_WEIGHT_BOLD: '700',  
  BACKGROUND_COLOR_LIGHT: '#f0f6f7',  
  CONTAINER_PADDING: 20  
};
```

Agora, no diretório `components/Home/`, crie um arquivo chamado `style.js`. Este irá utilizar o tema especificado para este aplicativo e será usado no componente `Home`.

```
import {StyleSheet} from 'react-native';  
import theme from '../style/theme.style';  
  
export default StyleSheet.create({  
  container: {  
    flex: 1,  
    paddingVertical: theme.CONTAINER_PADDING,  
    alignItems: 'center'  
  },  
  textAreaTitle: {  
    fontSize: theme.FONT_SIZE_MEDIUM,  

```

```

fontWeight: theme.FONT_WEIGHT_BOLD,
alignSelf: 'flex-start',
padding: 10
},
textArea: {
  flex: 1,
  fontSize: theme.FONT_SIZE_MEDIUM,
fontWeight: theme.FONT_WEIGHT_LIGHT,
padding: theme.CONTAINER_PADDING,
alignSelf: 'stretch',
overflow: 'scroll',
  backgroundColor: theme.BACKGROUND_COLOR_LIGHT
}
});

```

E, agora, altere o componente definido em Home/index.js para utilizar estes estilos.

```

import React, { useState } from "react";
import { View, Text, TextInput } from "react-native";
import styles from './style';

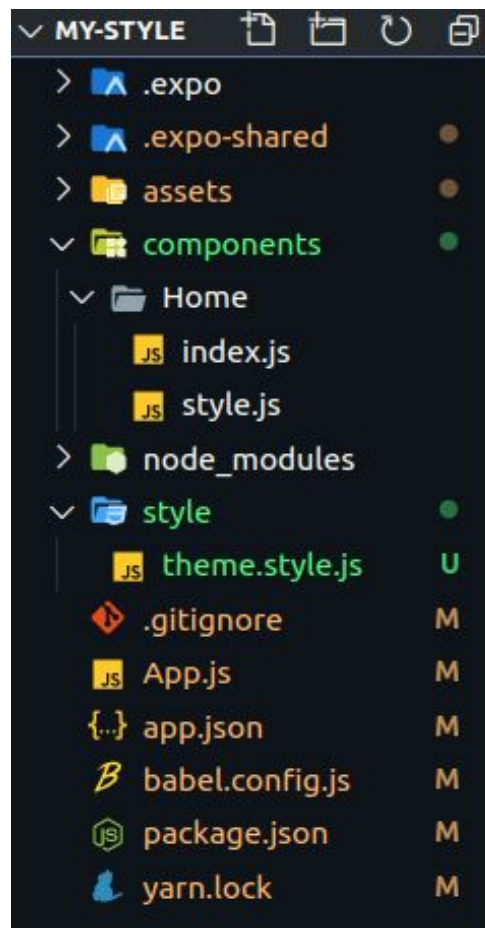
function Home() {
  const [state, setState] = useState({text: ''});
  return (
    <View style={styles.container}>
      <Text style={styles.textAreaTitle}> Digite sua anotação
aqui:</Text>
      <TextInput
        style={styles.textArea}
        multiline={true}
        numberOfLines={4}

```



```
      onChangeText={ (text) => setState ({text}) }  
      value={state.text} />  
    </View>  
  );  
}  
export default Home;
```

Veja como está a estrutura de nosso projeto:



E veja como está o projeto na tela:

Digite sua anotação aqui:

## 1.6 Estilos comuns

No React Native, cada componente é estilizado usando estilos embutidos (inline styles). Isso significa que se torna um pouco complicado compartilhar estilos da mesma forma que na web.

Na web, escrevemos uma classe CSS assim:

```
.btn {  
  padding: 10;  
  border: '1px solid black';  
}
```

Agora, se quisermos aplicar essa classe a dois `divs` diferentes, faremos o seguinte:

```
<div class='btn first-btn'>First button</div>  
<div class='btn second-btn'>Second button</div>
```

O mesmo é possível no React Native da seguinte maneira:

**arquivo styles.js**

```
import { StyleSheet } from 'react-native';
export default StyleSheet.create({
  btn: {
    padding: 10,
    borderWidth: 1
  },
  firstBtn:{
    ...
  },
  secondBtn:{
    ...
  }
});
```

Depois, adicionaremos estilos à nossa View:

**arquivo Home/index.js**

```
import React from "react";
import { View } from "react-native";
import styles from './styles.js';
function Home() {
  return (
    <View>
      <View style={[styles.btn, styles.firstBtn]}>First
button</View>
      <View style={[styles.btn, styles.secondBtn]}>Second
button</View>
    </View>
  );
}
```

Isso resolve o problema apenas se os objetos de estilo estiverem no mesmo componente, pois no React Native não importamos estilos de outros componentes (cada componente tem seu próprio estilo). Mas na web, poderíamos simplesmente reutilizar a classe em qualquer lugar (já que o CSS é global).

Para resolver o problema de estilos reutilizáveis no React Native, veja este outro arquivo chamado `style/common.style.js`. É onde escreveremos nossos estilos comuns do aplicativo. Portanto, se todos os botões em nosso aplicativo tiverem um estilo semelhante, podemos escrever um estilo com propriedades semelhantes dentro do `common.style.js`

#### arquivo `style/common.style.js`

```
import { StyleSheet } from 'react-native';

export default StyleSheet.create({

  btn: {

    padding: 10,

    borderWidth: 1

  }

});
```

E podemos apenas importá-lo em nossos arquivos de estilo de cada componente e reutilizá-lo diretamente desta forma:

#### arquivo `styles.js`

```
import { StyleSheet } from 'react-native';

import common from '../style/common.style.js';

export default StyleSheet.create({

  firstBtn:{

    ...common.btn,

    backgroundColor: 'blue'

  },

  secondBtn:{

    ...common.btn,

    backgroundColor: 'red'

  }

});
```

Depois, vamos adicionar os estilos à nossa View:

#### arquivo Home/index.js

```
import React from "react";
import { View } from "react-native";
import styles from './styles.js';

function Home() {
  return (
    <View>
      <View style={styles.firstBtn}>First button</View>
      <View style={styles.secondBtn}>Second button</View>
    </View>
  );
}

export default Home;
```

Desta forma, nosso arquivo de estilo comum nos fornecerá os estilos básicos que são comuns em todo o aplicativo e escreveremos estilos específicos de cada componente no arquivo de estilos dos componentes. Isso permite a reutilização significativa de estilo e evita a duplicação de código.

## 1.7 Integrando estilos comuns em nosso aplicativo

Antes de desenvolvermos estilos comuns em nosso projeto, vamos modificar um pouco nosso código.

Primeiro, vamos criar um diretório chamado TextArea e dentro criamos um arquivo chamado index.js com o seguinte código:

```
import React, { useState } from "react";
import { TextInput } from "react-native";
import styles from './style';

function TextArea() {
  const [state, setState] = useState({text: ''});
```

```

return (
  <TextInput
    style={styles.textArea}
    multiline={true}
    numberOfLines={4}
    onChangeText={(text) => setState({ text })}
    value={state.text}
  />
);
}
export default TextArea;

```

Agora, crie um arquivo de estilos em TextArea/style.js:

```

import {StyleSheet} from 'react-native';
import theme from '../../style/theme.style';

export default StyleSheet.create({
  textArea: {
    fontSize: theme.FONT_SIZE_MEDIUM,
    fontWeight: theme.FONT_WEIGHT_LIGHT,
    padding: theme.TEXT_INPUT_PADDING,
    backgroundColor: theme.BACKGROUND_COLOR_LIGHT,
    alignSelf: 'stretch',
    flex: 1
  }
});

```

Em seguida, altere o arquivo Home/index.js:

```

import React, { useState } from "react";
import { View, Text, TextInput } from "react-native";

```

```

import TextArea from '../TextArea';
import styles from './style';

function Home() {
  const [state, setTitle] = useState({title:''});
  return (
    <View style={styles.container}>
      <Text style={styles.titleHeading}>Título da anotação</Text>
      <TextInput style={styles.titleTextInput}
        onChangeText={({title}) => setTitle({title})}
        value={state.title} />
      <Text style={styles.textAreaTitle}>Digite sua anotação
aqui:</Text>
      <TextArea />
    </View>
  );
}

export default Home;

```

Agora, vamos alterar o arquivo Home/style.js:

```

import {StyleSheet} from 'react-native';
import theme from '../../style/theme.style';
export default StyleSheet.create({
  container: {
    flex: 1,
    paddingVertical: theme.CONTAINER_PADDING,
    alignItems: 'center'
  },
  titleHeading: {
    fontSize: theme.FONT_SIZE_MEDIUM,
    alignSelf: 'flex-start',

```

```

padding: 10,
fontWeight: theme.FONT_WEIGHT_BOLD,
},
titleTextInput: {
padding: theme.TEXT_INPUT_PADDING,
backgroundColor: theme.BACKGROUND_COLOR_LIGHT,
alignSelf: 'stretch'
},
textAreaTitle: {
fontSize: theme.FONT_SIZE_MEDIUM,
alignSelf: 'flex-start',
padding: 10,
fontWeight: theme.FONT_WEIGHT_LIGHT,
fontStyle: 'italic'
}
});

```

E, finalmente, vamos alterar o arquivo `styles/theme.style.js`:

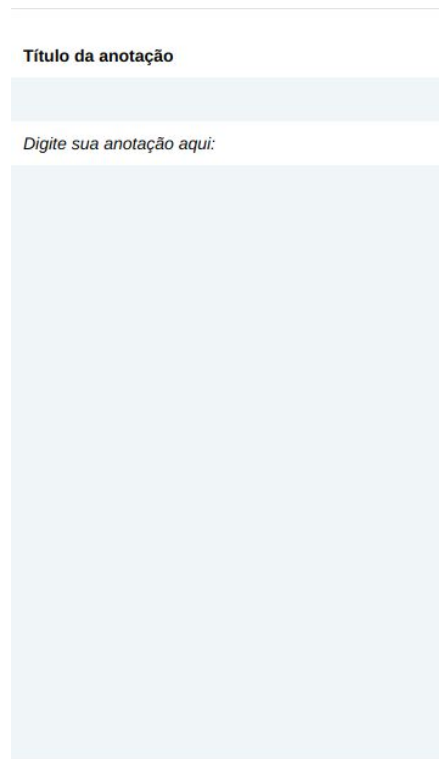
```

export default {
PRIMARY_COLOR: '#2aabb8',
FONT_SIZE_SMALL: 12,
FONT_SIZE_MEDIUM: 14,
FONT_SIZE_LARGE: 16,
FONT_WEIGHT_LIGHT: '200',
FONT_WEIGHT_MEDIUM: '500',
FONT_WEIGHT_BOLD: '700',
BACKGROUND_COLOR_LIGHT: '#f0f6f7',
CONTAINER_PADDING: 20,
TEXT_INPUT_PADDING: 10
};

```



Nosso aplicativo agora deve ter a seguinte aparência:



The mockup shows a simple note-taking interface. At the top, there is a light blue header bar with the text 'Título da anotação' in a bold, dark font. Below the header, there is a light blue rectangular area for the note content. Above this area, the text 'Digite sua anotação aqui:' is displayed in a smaller, italicized font. The entire interface is set against a white background.

Se você notar, embora tenhamos um arquivo de tema, nosso código de estilo possui muitos códigos duplicados. Isso ocorre principalmente porque estamos repetindo nosso estilo para a entrada de texto e também para o título.

A solução para este problema faremos uso de estilos comuns. Assim, para resolver isso, vamos criar o arquivo `style/common.style.js`:

```
import theme from './theme.style';

export const headingText = {
  fontSize: theme.FONT_SIZE_MEDIUM,
  alignSelf: 'flex-start',
  padding: 10,
  fontWeight: theme.FONT_WEIGHT_BOLD,
};

export const textInput = {
  padding: theme.TEXT_INPUT_PADDING,
  backgroundColor: theme.BACKGROUND_COLOR_LIGHT,
  alignSelf: 'stretch'
};
```

E modifique nossos arquivos de estilo de componente para incluir o common.style.js

#### arquivo Home/style.js

```
import {StyleSheet} from 'react-native';
import theme from '../../style/theme.style';
import {headingText, textInput} from
'../../style/common.style';
export default StyleSheet.create({
  container: {
    flex: 1,
    paddingVertical: theme.CONTAINER_PADDING,
    alignItems: 'center'
  },
  titleHeading: {
    ...headingText
  },
  titleTextInput: {
    ...textInput
  },
  textAreaTitle: {
    ...headingText,
    fontWeight: theme.FONT_WEIGHT_LIGHT,
    fontStyle: 'italic'
  },
});
```

#### arquivo TextArea/style.js

```
import { StyleSheet } from "react-native";
import theme from "../../style/theme.style";
import {textInput} from '../../style/common.style';
```

```
export default StyleSheet.create({
  textArea: {
    ...textInput,
    flex: 1,
    fontSize: theme.FONT_SIZE_MEDIUM,
    fontWeight: theme.FONT_WEIGHT_LIGHT,
  },
});
```

Você pode observar que nosso código de estilo parece muito mais conciso. Veja que estamos reajustando os estilos de componentes semelhantes com pequenas alterações.

Veja o resultado na tela:

**Título da anotação**

*Digite sua anotação aqui:*