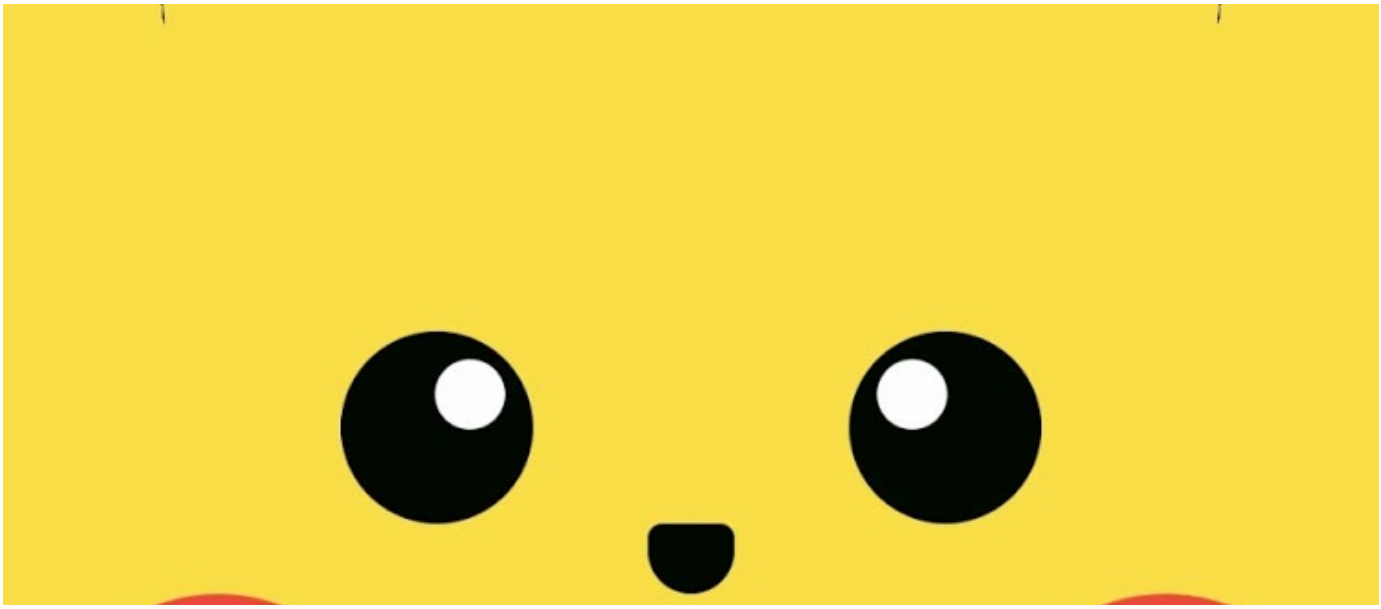
Artigos > Mobile

React Native - Utilizando styled-components

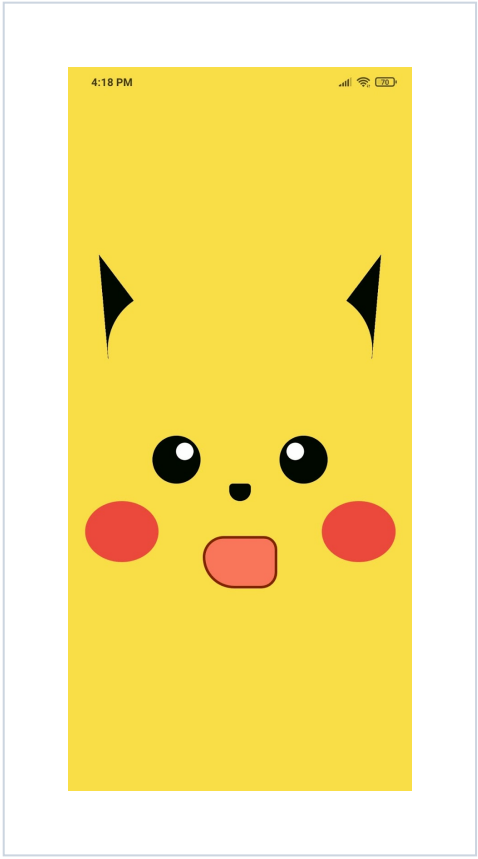


Natalia Kelim Thiel
28/04/2021

Eis que chega o momento em que você quer fazer um app baseado em uma aplicação web ou apenas criar um app novo, utilizando um padrão de estilos de fácil manutenção e organização, sem ter *tags style* em cada componente que você usa. Com a biblioteca styled-components podemos escrever estilos em um formato muito semelhante ao CSS e criar componentes com base nos estilos sem a necessidade de poluir a estrutura da nossa aplicação. Mas como utilizar styled-components na minha aplicação?

Projeto

Para exemplificar, neste artigo vamos usar a tela abaixo que já está desenvolvida com a forma padrão de estilos. Fique à vontade para clonar o projeto, instalar de acordo com o readme e executar o projeto (você **pode** rodá-lo sem Android Studio e XCode).



Aplicando styled-components

Vamos começar a transformar nossos estilos em styled-components pela boca. Mas, calma lá, não estamos esquecendo algo? Sim, falta instalar a biblioteca! Para isso, executamos o seguinte comando, dentro da pasta do projeto, e ainda temos que rodar o projeto novamente:

```
npm install --save styled-components
```

Agora, podemos usar styled-components no nosso projeto! Vamos ver como está o componente da boca:

Antigo `src/Pikachu/Boca/index.js`

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

import estilos from './estilos';

export default function Boca() {
    return <View style={estilos.boca} />
}
```

Antigo `src/Pikachu/Boca/estilos.js`

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

export default StyleSheet.create({
    boca: {
        width: 85,
        height: 60,
        marginTop: -30,
        backgroundColor: "#F18061",
        borderWidth: 3,
        borderColor: "#7B2B09",
        borderTopStartRadius: 40,
        borderTopEndRadius: 25,
        borderBottomEndRadius: 30,
        borderBottomStartRadius: 60,
    }
});
```

Podemos resumir esse código todo em apenas um arquivo com styled-components, que pode ser o `index.js` :

Novo `src/Pikachu/Boca/index.js`

```
import styled from 'styled-components';

const Boca = styled.View`
    width: 85px;
    height: 60px;
    margin-top: -30px;
    background-color: #F18061;
    border: 3px solid #7B2B09;
    border-top-start-radius: 40px;
    border-top-end-radius: 25px;
    border-bottom-end-radius: 30px;
    border-bottom-start-radius: 60px;
`;

export default Boca;
```

Como você pode ver, importamos os styled-components na variável `styled` e usamos o objeto `View` dentro dele para criarmos um componente do tipo view customizado. No lugar da `View` poderíamos usar qualquer outro componente **React Native**. Então colocamos nosso "CSS" dentro de crases. Mas temos que fazer algumas mudanças em relação aos estilos normais, que são:

- Mudar as vírgulas por ponto e vírgulas `;` .
- Adicionar as unidades `px` após os números.
- Remover as aspas dos hexadecimais de cor.
- Trocar a nomenclatura adicionando o traço em vez da primeira letra maiúscula.
- Não é obrigatório, mas podemos também reescrever `borderWidth` e `borderColor` em uma propriedade unificada `border` . Pensando bem, seria legal fazer o mesmo com `border-radius` , né? Porém, por conta de problemas com a *tag* `Image` , essa propriedade não está habilitada e deve ser possível usá-la apenas na versão 6 do styled-components.

Vamos fazer o mesmo com o nariz e as bochechas. Como eles também são componentes `View` estilizados, podemos apenas retornar seus respectivos *index*, não sendo necessário mais um arquivo de estilos. Para o nariz, basta converter da mesma forma que fizemos antes, de `estilos.js` para `index.js` .

Antigo `src/Pikachu/Nariz/estilos.js`

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

export default StyleSheet.create({
    nariz: {
        width: 25,
        height: 20,
        backgroundColor: "#000200",
        borderTopStartRadius: 10,
        borderTopEndRadius: 10,
        borderBottomStartRadius: 30,
        borderBottomEndRadius: 30,
    }
});
```

Novo `src/Pikachu/Nariz/index.js`

```
import styled from 'styled-components';

const Nariz = styled.View`
    width: 25px;
    height: 20px;
    backgroundColor: #000200;
    border-top-start-radius: 10px;
    border-top-end-radius: 10px;
    border-bottom-start-radius: 30px;
    border-bottom-end-radius: 30px;
`;

export default Nariz;
```

Já com as bochechas, temos uma propriedade diferente: a `transform`. Nos estilos com React Native, temos de usar esse formato não tão interessante de array com objeto.

Com styled-components, já podemos fazer em forma de CSS mesmo, veja abaixo.

Antigo `src/Pikachu/Bochecha/estilos.js`

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

export default StyleSheet.create({
    bochecha: {
        width: 70,
        height: 70,
        backgroundColor: "#E15B42",
        borderRadius: 35,
        transform: [{scaleX: 1.2}]
    }
});
```

Novo `src/Pikachu/Bochecha/index.js`

```
import styled from 'styled-components';

const Bochecha = styled.View`
    width: 70px;
    height: 70px;
    backgroundColor: #E15B42;
    border-radius: 35px;
    transform: scaleX(1.2);
`;

export default Bochecha;
```

Herdando Estilos

Agora vamos incrementar um pouco as coisas, pois nos olhos temos dois componentes `View` e estilos externos sendo aplicados. Os estilos servem para fazer o espelhamento horizontal do olho direito e conseguirmos os dois brilhinhos na parte central do rosto.

Antigo `src/Pikachu/Olho/index.js`

```
import React from 'react';
import { View } from 'react-native';
import estilos from './estilos'

export default function Olho({ estilosExtra }) {
    return <View style={[estilos.olho, estilosExtra]} >
        <View style={estilos.brilho} />
    </View>
}
```

Mas vamos primeiro olhar para a estrutura do componente. Como temos a `View` interna, não podemos retornar apenas a view estilizada como temos feito. Precisamos manter essa estrutura e estilizar as duas `View` s. Então, neste caso, nosso `estilos.js` continuará existindo, porém, podemos mudar para a estrutura dos styled-components e retornar mais componentes estilizados.

Antigo `src/Pikachu/Olho/estilos.js`

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

export default StyleSheet.create({
    olho: {
        backgroundColor: "#000200",
        width: 55,
        height: 55,
        borderRadius: 28,
    },
    brilho: {
        backgroundColor: "#FEFEFE",
```

```
        width: 20,
        height: 20,
        borderRadius: 10,
        marginTop: 8,
        marginLeft: 27,
    }
});
```

Novo `src/Pikachu/Olho/estilos.js`

```
import styled from 'styled-components';

export const OlhoExterno = styled.View`
    background-color: #000200;
    width: 55px;
    height: 55px;
    border-radius: 28px;
`;

export const Brilho = styled.View`
    background-color: #FEFEFE;
    width: 20px;
    height: 20px;
    border-radius: 10px;
    margin: 8px 0 0 27px;
`;
```

Assim também simplificamos as margens em apenas uma linha de `margin`, como em CSS, e renomeamos o `olho` para `OlhoExterno`, pois `Olho` já é o nome do nosso componente principal. Então, podemos salvar e... acontece um erro. Isso ocorreu porque ainda não alteramos nosso `index.js` para utilizar os nossos novos componentes estilizados.

Novo `src/Pikachu/Olho/index.js`

```
import React from 'react';

import { OlhoExterno, Brilho } from './estilos'

export default function Olho({ estilosExtra }) {
    return <OlhoExterno style={estilosExtra} >
        <Brilho />
    </OlhoExterno>
}
```

Se você estava se perguntando: “Como vou aplicar styled-components na minha aplicação que já tem um tamanho considerável? É impossível alterar meus 500 componentes todos de uma vez”. Uma resposta é fazer aos poucos. Nesse exemplo do olho, estamos recebendo ainda os estilos padrão do React Native, e eles ainda funcionam. Mas como essa aplicação é mais simples, vamos transformar até os `estilosExternos` em styled-components. Para isso, utilizamos a herança de componentes. Se olharmos os arquivos `index.js` e `estilos.js` principais temos o segundo olho sendo chamado com os estilos.

Olhos em `src/Pikachu/index.js`

```
…
<View style={estilos.olhos}>
        <Olho />
    <Olho estilosExtra={estilos.olhoDireito} />
</View>
…
```

Estilos dos olhos em `src/Pikachu/estilos.js`

```
…
olhoDireito: {
        transform: [{scaleX: -1}],
```

```
},
...
```

Vamos transformar, então, o segundo olho em um componente dentro dos estilos do index principal, mas que herda o componente de olho original. Também podemos remover o estilo `olhoDireito` dos estilos React Native.

Novos estilos em `src/Pikachu/estilos.js`

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

// Início do código novo
import styled from 'styled-components';
import Olho from './Olho';

export const OlhoDireito = styled(Olho)`
    transform: scaleX(-1);
`;
// Fim do código novo

export default StyleSheet.create({
...
```

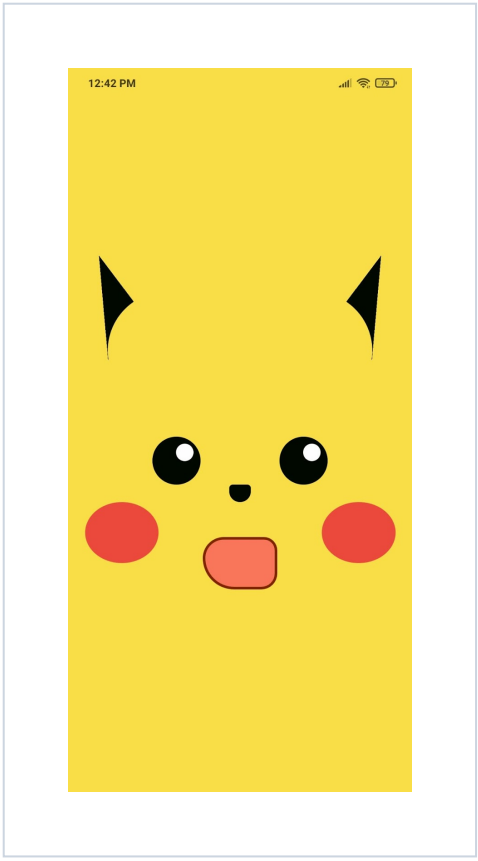
Passando dessa forma um outro componente ou styled-component para a função `styled`, podemos criar um novo styled-component que herda a estrutura e os estilos dele. Então, podemos usar nosso novo componente que herda `Olho` e o inverte no `index.js`.

Novo olho em `src/Pikachu/index.js`

```
...
import estilos, { OlhoDireito } from './estilos';
…
<View style={estilos.olhos}>
        <Olho />
```

```
        <OlhoDireito />
</View>
...
```

Então salvamos, mas não fica bem como esperamos, pois a herança não está funcionando... Os dois olhos estão iguais, não invertidos.



Isso acontece porque o `Olho` é um componente criado da forma tradicional no React Native, que usa styled-components dentro dele. Portanto, precisamos permitir que os nossos styled-components internos recebam as alterações do nosso `OlhoDireito` . Caso estivéssemos tentando aplicar herança na boca, por exemplo, não teríamos este problema, pois a boca já é um styled-component.

Permitindo herança no `src/Pikachu/Olho/index.js`

```
import React from 'react';
import { OlhoExterno, Brilho } from './estilos'

export default function Olho({ style }) {
    return <OlhoExterno style={style} >
        <Brilho />
```

```
    </OlhoExterno>
}
```

Agora, passando a propriedade style para o `OlhoExterno`, podemos rodar nossa aplicação e a herança funciona. Também finalizamos a aplicação dos styled-components no `Olho`.

Acesso às propriedades

A estrutura da orelha é composta por um triângulo e um círculo que cobre parte do triângulo, fazendo o efeito de arredondamento no lado inferior. Na implementação das orelhas também fazemos a inversão, porém, de uma forma diferente: passamos a propriedade `direita` e o componente faz a inversão internamente.

Antigo `src/Pikachu/Orelha/index.js`

```
import React from 'react';
import { View } from 'react-native';
import funcaoEstilos from './estilos';

export default function Orelha({ direita = false }) {
    const estilos = funcaoEstilos(direita);

    return <View style={estilos.orelha}>
        <View style={estilos.ponta} />
        <View style={estilos.marca} />
    </View>
}
```

Então, utilizamos nos estilos uma função para pegar este valor e retornar o `StyleSheet` correspondente.

Antigo `src/Pikachu/Orelha/estilos.js`

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

export default function (direita) {
    return StyleSheet.create({
        orelha: {
        transform: [{rotate: direita ? "5deg" : "-5deg" }, {scaleX: direi
        },
        ponta: {
        width: 0,
        height: 0,
        backgroundColor: "transparent",
        borderStyle: "solid",
        borderRightWidth: 120,
        borderTopWidth: 75,
        borderRightColor: "transparent",
        borderTopColor: "#000200",
        transform: [{rotate: "270deg"}]
        },
        marca: {
        width: 135,
        height: 135,
        marginTop: -50,
        marginLeft: 23,
        backgroundColor: "#FCD458",
        borderRadius: 77,
        }
    });
}
```

Quando transformamos para styled-components temos uma estrutura mais simples para acessar as propriedades.

Novo `src/Pikachu/Orelha/estilos.js`

```
export const OrelhaExterna = styled.View`
    transform: rotate(-5deg) scaleX(${({ direita }) => direita ? -1 : 1})
`;

export const Ponta = styled.View`
    width: 0;
    height: 0;
    background-color: transparent;
    border-style: solid;
    border-right-width: 120px;
    border-top-width: 75px;
    border-right-color: transparent;
    border-top-color: #000200;
    transform: rotate(270deg);
`;

export const Marca = styled.View`
    width: 135px;
    height: 135px;
    margin: -50px 0 0 23px;
    border-radius: 77px;
    background-color: #FCD458;
`;
```

Antes de falar das propriedades passadas, tem duas coisas diferentes neste exemplo:

1. Invertemos os parâmetros do `transform`, passando `rotate` primeiro, depois `scaleX`. Precisamos fazer isso porque o styled-components inverte a ordem de processamento do `transform`. O que queremos, no fim das contas, é que primeiro seja aplicado o `rotate` para deixar a orelha um pouco inclinada e depois `scaleX` para que ela fique espelhada. Se o inverso acontecer, primeiro vamos inverter e depois rotacionar, mas será rotacionada para o lado errado, fazendo as orelhas ficarem paralelas, e não apontando para lados opostos.
2. A segunda coisa é novamente o `border`. Também não conseguimos simplificar o border em menos linhas, pois isso não é suportado. Então, apenas convertemos da forma que estava. Agora vamos mostrar como acessar as propriedades. Podemos abrir um trecho de código dentro da string de estilos com `${}`, onde podemos inserir uma variável ou constante que o código tenha

acesso diretamente (por exemplo, `${corPrimaria}`) ou criar como uma função desta forma: `${(props) => props.direita}`. Quando utilizamos em forma de função conseguimos acessar as propriedades passadas para o componente, como `direita` no nosso caso. Ainda no exemplo fazemos uma desconstrução acessando diretamente a propriedade `direita` dentro de `props`, então, não precisamos chamar `props.` toda vez que queremos usar a variável direita.

Desafio

Se você já acompanhou até aqui e quer treinar um pouco mais, que tal começar por aplicar styled-components no componente Pikachu que ficou faltando? Caso tenha perdido alguma coisa até aqui, você encontra o projeto neste ponto na branch `styled`.

Conclusão

Neste artigo você pôde aprender os conceitos básicos de styled-components, transformar estilos padrão em styled-components, utilizar herança e acessar propriedades. Além, é claro, de poder baixar um modelo de Pikachu feito apenas com estilos. Também vimos algumas características que diferem os estilos padrão dos styled-components e do CSS, quando aplicado no React. Abaixo, um resumo dessas diferenças.

- Precisamos remover as aspas dos estilos padrão, trocar as vírgulas por ponto e vírgula e adicionar " `px` " no fim dos números.
- Renomeamos nossas propriedades para ficarem com " `-` " (hífen) e não letra maiúscula entre as palavras, por exemplo, `backgroundColor` vira `background-color`.
- Algumas propriedades, como `border-radius` e `border`, suportam apenas um valor como parâmetro, sendo assim não podemos simplificar vários lados em uma única linha.
- Caso a ordem do `transform` importe, ela deve ser invertida, pois o styled-components executa de trás para frente.
- Esta não vimos no artigo: se comparar com CSS, o suporte a media queries e keyframes é inexistente com styled-components.

Caso queira acessar o projeto finalizado com todos os estilos no formato styled-components, você pode baixá-lo na branch `completo`.

Referências

- styled-components
- Curso – React: Abstraindo seu CSS com Styled Components
- React: componentes com Styled Components

Confira neste artigo:

Natalia Kelim Thiel

Natalia é programadora e instrutora. Se apaixonou pela programação a primeira vista em 2013 e desde então vem trabalhando em diversas tecnologias no front-end, back-end, mobile e games.

Artigo Anterior

Flutter - Como configurar o ambiente de desenvolvimento

Próximo Artigo

Flutter - Null Safety

Veja outros artigos sobre
Mobile

Quer mergulhar em tecnologia e aprendizagem?

Receba a newsletter que o nosso CEO escreve pessoalmente, com insights do mercado de trabalho, ciência e desenvolvimento de software

Escreva seu email

ME INSCREVA

Nossas redes e apps

Institucional

Sobre nós

Trabalhe conosco

Para Empresas

A Alura

Como Funciona

Todos os cursos

Depoimentos

Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de DevOps

AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design

Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile

React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos

Cursos de Inovação & Gestão

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas