Desenvolvimento para Dispositivos Móveis

Fausto Toloi

Revisão React JSX

JSX em React

Considere esta declaração de variável:

```
const element = <Text>Hello, world!</Text>;
```

// Essa sintaxe de tag curiosa pois não é uma string nem HTML.

JSX é uma extensão de sintaxe para JavaScript.

JSX produz “elementos” React. Vamos explorar a renderização deles para o DOM.

Revisão React JSX

Incorporando Expressões em JSX do React

```jsx
export function componenteExemplo() {
  const name = 'Nome Maravilhoso da Silva';
  return (
    <Text>Nome {{ name }}</Text>
  )
}
```

Revisão React JSX

Incorporando Expressões em JSX do React

incorporamos o resultado da chamada de uma função JavaScript, formatName(user), em um <h1>elemento.

```
export function HelloUser() {

  const user = {
    primeiroNome: 'Luiz Carlos',
    sobrenome: 'Silva',
  }

  function formatName(user) {
    return user.primeiroNome + ' ' + user.sobrenome;
  }

  return (
    <Text>Olá, {formatName(user)}!</Text>
  )

}
```

Revisão React JSX

JSX representa objetos

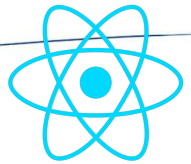
Babel compila JSX para React.createElement() chamadas.

Estes dois exemplos são idênticos:

```jsx
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

```jsx
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```



Componentes React de função e de Classe

A maneira mais simples de definir um componente é escrever uma função JavaScript:

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

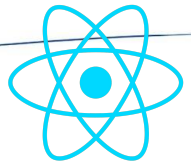
=

```jsx
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Sempre comece os nomes dos componentes com **uma letra maiúscula.**

O React trata os componentes que começam com letras minúsculas como tags DOM.

Por exemplo, `<div />` representa uma tag HTML div, mas `<Welcome />` representa um componente e Welcome precisa estar no no escopo.



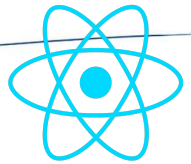
Renderizando um componente

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

React

```jsx
export default function App() {
  function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }

  const element = <Welcome name="Sara" />;

  return (
    <View>
      <Welcome />
      {element}
    </View>
  );
}
```

App.js - React Native



Renderizando um componente

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

```
export default function App() {
  function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }
  return (
    <View>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </View>
  );
}
```

App.js - React Native

Boas práticas em Componentes React usando HTML

Fragmentando componentes

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

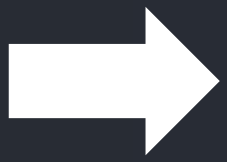
```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}
```

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

Boas práticas em Componentes em React

Fragmentando componentes

```jsx
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```



```jsx
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Revisão React JSX

Criando um componente Relógio

Componente Relógio usando o HTML para React, imagine div como View e h1,h2 como Text.

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

function Relogio() {
  return (
    <div>
      <h1>Veja as horas</h1>
      <h2>Hora atual: {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
}
```

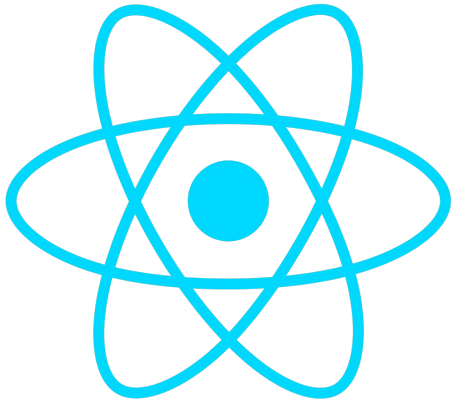
Introdução ao React Native

- Visão geral do React Native e suas vantagens:
  - Como o React Native pode ser uma solução para desenvolvimento de aplicativos multiplataforma, com uma única base de código e interface nativa.

React Native

O **React Native** combina as melhores partes do desenvolvimento nativo com o **React**.

Podemos compilar em projetos Android e iOS.



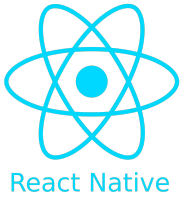
Vantagens:

- Acessar a interface e os recursos nativos do Android e iOS utilizando JavaScript;
- O código produzido é semelhante ao React para Web;
- Possui a base de conhecimento compartilhada entre o desenvolvimento mobile e front-end;
- Todo código desenvolvido é convertido para a linguagem nativa do sistema operacional;
- Com o React Native conseguimos desenvolver aplicações para Android e iOS utilizando um código único;
- Por ser multiplataforma, podemos desenvolver aplicações com React Native utilizando qualquer sistema operacional (Windows, macOS ou Linux).

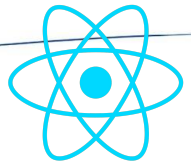
Senac

Introdução ao React Native

- Diferenças entre React Native e desenvolvimento nativo:
  - Principais diferenças entre o React Native e o desenvolvimento nativo, destacando as vantagens e desvantagens de cada abordagem.



Multiplataforma	Os aplicativos funcionam tanto no iOS quanto no Android	O aplicativo é feito para uma plataforma específica
Desenvolvedores	Desenvolvedores com uma sólida experiência em desenvolvimento web	Desenvolvedores com um forte conhecimento da plataforma
Tempo de construção	Rápido	Devagar
Formulários	Simples	Complexo
Suporte de longo termo	Não recomendado, pois, o framework pode ser descontinuado	Recomendado para investimentos de longo prazo
Recursos	Melhor para pequenas equipes de desenvolvimento com recursos limitados	Melhor para desenvolvedores experientes com recursos adequados



Componentes básicos do React Native

Visão geral dos componentes nativos do React Native:

- Componentes nativos do React Native
  - Text, Image, View, TextInput e ScrollView, destacando suas principais propriedades e funcionalidades.

Criação de componentes personalizados:

- Como criar componentes personalizados a partir de componentes nativos, usando herança ou composição.

Exemplos de componentes básicos: Text, Image, View, StyleSheet

Link para documentação dos componentes nativos do React Native: https://reactnative.dev/docs/components-and-apis

Link para documentação de criação de componentes personalizados: https://reactnative.dev/docs/custom-components

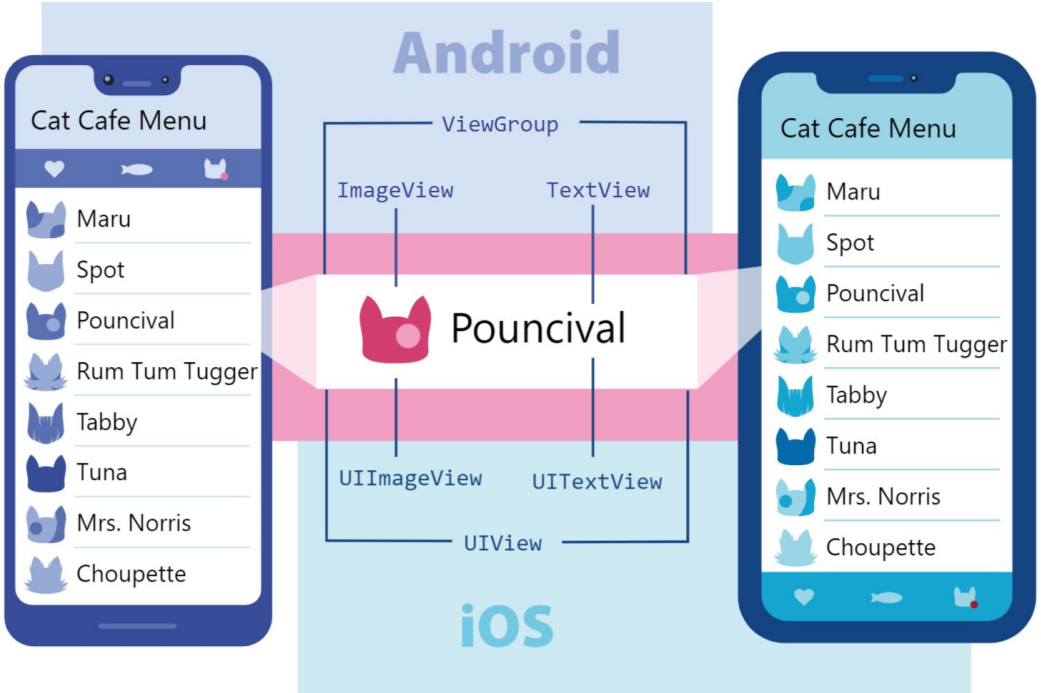
Link para documentação de estilização com StyleSheet: https://reactnative.dev/docs/style

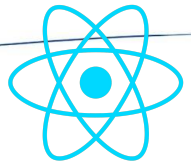
Senac

React Native

Componentes - View

No desenvolvimento Android e iOS, uma **View** é o bloco de interface do usuário: um pequeno elemento retangular na tela que pode ser usado para exibir texto, imagens ou responder à entrada do usuário. Mesmo os menores elementos visuais de um aplicativo, como uma linha de texto ou um botão, são tipos de **View** . Alguns tipos de **View** podem conter outras visualizações.

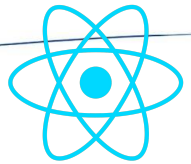




View, State, Props, e Style

A **View** é um componente elementar do **React Native** para construir uma interface de usuário. É um contêiner que suporta layout com flexbox, estilo, touch e controles de acessibilidade. Ele desenha diretamente para a visualização nativa do dispositivo.

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

const ViewBoxesWithColorAndText = () => {
  return (
    <View
      style={{
        flexDirection: 'row',
        height: 100,
        padding: 20,
      }}>
      <View style={{backgroundColor: 'blue', flex: 0.3}} />
      <View style={{backgroundColor: 'red', flex: 0.5}} />
      <Text>Hello World!</Text>
    </View>
  );
};

export default ViewBoxesWithColorAndText;
```

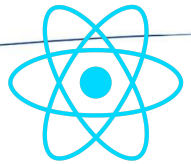


View, **State**, Props, e Style

Existem dois tipos de dados que controlam um componente no React Native: **props** e **state**. Para dados que vão mudar no futuro, usamos state. O estado contém os dados ou informações sobre o componente. Ele determina o comportamento do componente e como ele será renderizado.

- O objeto “state” contém os dados que são renderizados na tela
- Existem dois tipos de dados que controlam um componente: **props** e **state**. **props** são definidos pelo pai e são fixos durante todo o ciclo de vida de um componente. Para **dados** que serão alterados temos de usar **state**.
- Em geral, você deve inicializar state no construtor e, em seguida, chamá-lo **setState** quando quiser alterá-lo.
- Exemplo no próximo slide.

```tsx
import React, {useState, useEffect} from 'react';
import {Text, View} from 'react-native';

type BlinkProps = {
  text: string;
};

const Blink = (props: BlinkProps) => {
  const [isShowingText, setIsShowingText] = useState(true);

  useEffect(() => {
    const toggle = setInterval(() => {
      setIsShowingText(!isShowingText);
    }, 1000);

    return () => clearInterval(toggle);
  });

  if (!isShowingText) {
    return null;
  }

  return <Text>{props.text}</Text>;
};

const BlinkApp = () => {
  return (
    <View style={{marginTop: 50}}>
      <Blink text="I love to blink" />
      <Blink text="Yes blinking is so great" />
      <Blink text="Why did they ever take this out of HTML" />
      <Blink text="Look at me look at me look at me" />
    </View>
  );
};

export default BlinkApp;
```

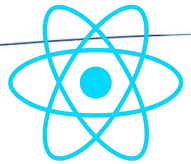


States e Hooks

O useState é um Hook de função que recebe um argumento, este argumento é o estado inicial do **state**, e retorna dois valores:

- O estado atual, ou seja o valor definido;
- Uma função que pode ser usada para atualizar o estado.

```
import React, { useState } from 'react'

function UserComponent() {
  const [name, setName] = useState('João, valor inicial')
}
```



Atribuição via desestruturação

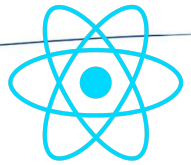
A sintaxe de atribuição via desestruturação (**destructuring assignment**) é uma expressão JavaScript que possibilita extrair dados de **arrays** ou **objetos** em variáveis distintas.

```
var foo = ["one", "two", "three"];

var [one, two, three] = foo;
console.log(one); // "one"
console.log(two); // "two"
console.log(three); // "three"
```

[ ] array

```
var {a=10, b=5} = {a: 3};

console.log(a); // 3
console.log(b); // 5
```

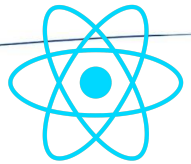
{ } objeto



View, State, **Props**, e Style

- Props é a abreviação de Propriedades. Um componente pode ser customizado no momento de ser usado utilizando parâmetros. Estes parâmetros são chamados de props. Props são passados de um container para outro como meio de transmitir dados entre eles.

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

const Bananas = () => {
  let pic = {
    uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg',
  };
  return (
    <Image source={pic} style={{width: 193, height: 110, marginTop: 50}} />
  );
};

export default Bananas;
```



View, State, **Props**, e **Style**

O **React Native** usa **JavaScript** para **estilizar** o aplicativo. Todos os componentes principais usam uma **prop** chamado “style”. Os nomes e valores de estilo são semelhantes ao CSS que funciona para a web. Para estilizar nossos componentes, podemos usar estilo inline ou usar StyleSheet, que é um componente React Native.

O estilo embutido é usado para o texto "estilo embutido"

Exemplo no próximo slide

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

const LotsOfStyles = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.red}>just red</Text>
      <Text style={styles.bigBlue}>just bigBlue</Text>
      <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
      <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 50,
  },
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});

export default LotsOfStyles;
```

Expo

O **Expo** é uma ferramenta utilizada no desenvolvimento mobile com **React Native** que permite o fácil acesso às **API's nativas** do dispositivo.

Vantagens:

- Fácil de usar
- rápido implantar um projeto
- Bibliotecas e documentação
- Desenvolvimento ativo

Desvantagens:

- Pula parte importante da implementação de um app
- Usa APIs internos para acessar recursos do celular

Senac

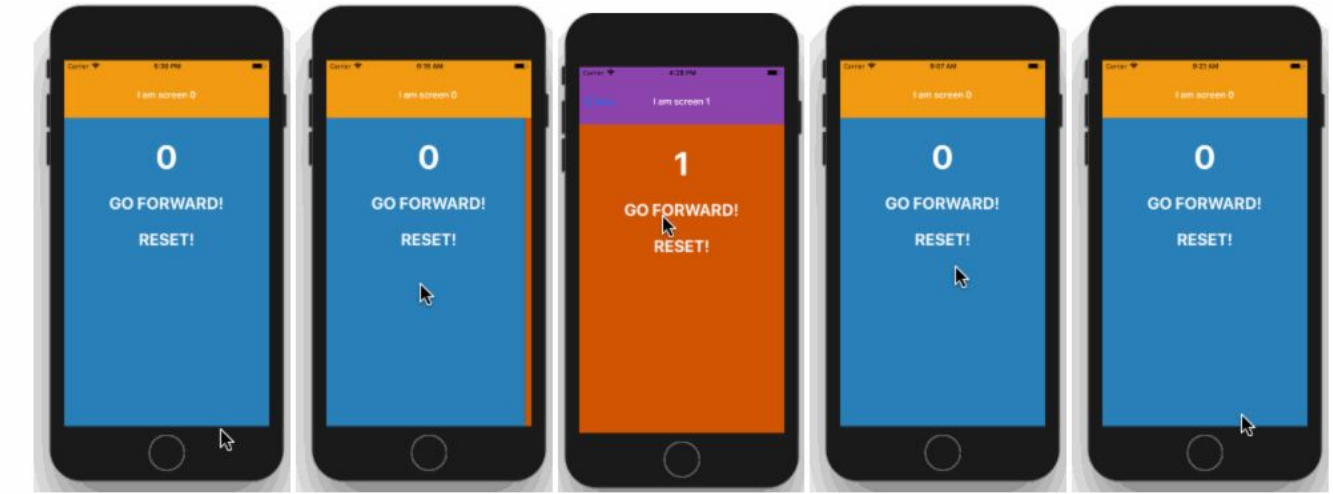
Expo

```
npm install --global expo-cli
expo init my-app --template blank
cd my-app
yarn add @types/react-native
expo start:web
```

Auxilia o VS Code a importar corretamente as bibliotecas [1]

React Navigation

- É uma biblioteca de navegação popular no ecossistema **React Native** e também é mantida pela **Expo**.
- Inclui suporte para padrões de navegação comuns, como pilhas, guias e gavetas.
- Ele também foi desenvolvido para ser **personalizável**, para que você possa obter qualquer padrão de navegação que desejar, mesmo que não esteja integrado à biblioteca.
- Ele suporta o **uso das APIs** nativas da plataforma via ***createNativeStackNavigator***, referido no React Native como "navegação nativa".



React Navigation

Fácil de usar

Desenvolve de **forma rápida navegação** integrada ao SO.

Personalizável

Usando **JavaScript**, é possível personalizar o **React Navigation**.

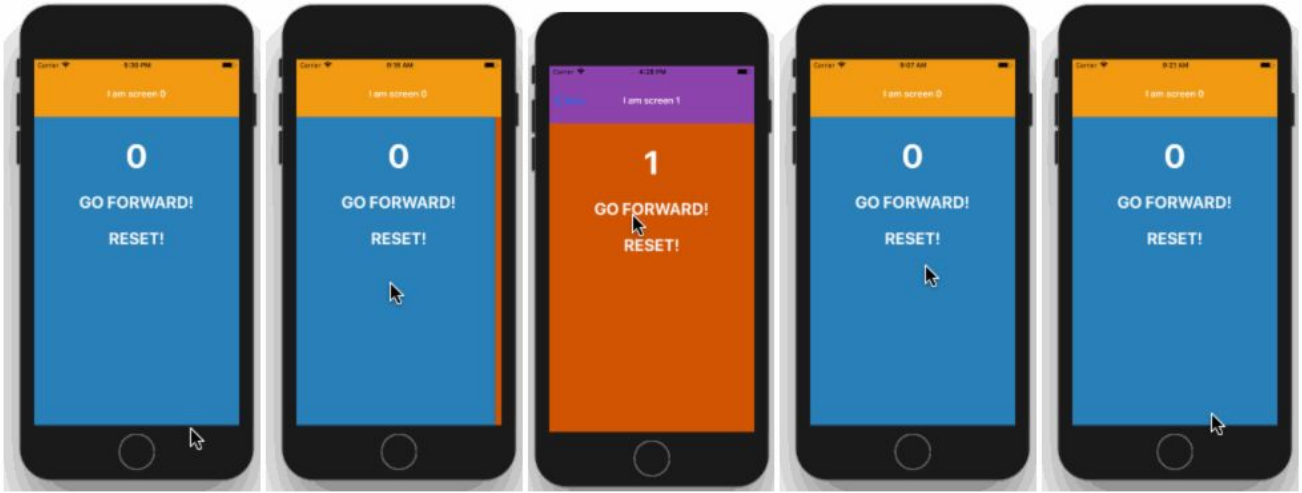
Componentes criados para iOS e Android

Aparência **específica da plataforma** com animações e gestos.

Plataforma extensível

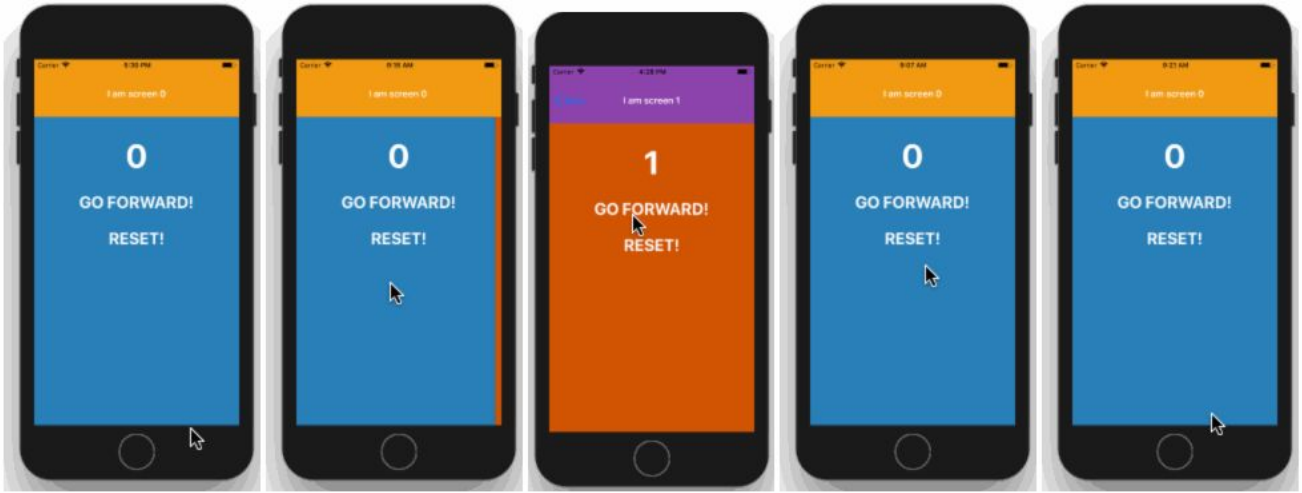
O **React Navigation** é extensível em todas as camadas - você pode escrever seus próprios navegadores ou até mesmo **substituir a API** voltada para o usuário.

Ver Vídeo



Pré-requisitos

- **react-native**>= 0,63,0
- **expo**>= 41 (se você usar Expo)
- **typescript**>= 4.1.0 (se você usa TypeScript)



React Navigation

```
yarn add
  @react-navigation/native
  @react-navigation/native-stack
  @react-navigation/drawer
  @react-navigation/bottom-tabs
  @react-navigation/material-bottom-tabs
```

Importaremos o **NavigationContainer o** componente que gerencia nossa árvore de navegação e contém o estado de navegação. Este componente deve envolver toda a estrutura do App.

Importaremos a função **createNativeStackNavigator** que retorna um objeto contendo 2 propriedades: **Screen** e **Navigator**.

Importaremos a função **createDrawerNavigator** que retorna um objeto contendo 2 propriedades: **Screen** e **Navigator**.

Importaremos a função **createBottomTabNavigator**

native-stack, **drawer, bottom-tabs** retornam um objeto contendo 2 propriedades: Screen e Navigator.

Navigator deve conter elementos **Screen** como filhos para definir a configuração das rotas.

React Navigation - Bibliotecas adicionais

expo install / yarn add / npm i

```
yarn add react-native-screens
yarn add react-native-safe-area-context
yarn add react-native-gesture-handler
yarn add react-native-reanimated
```

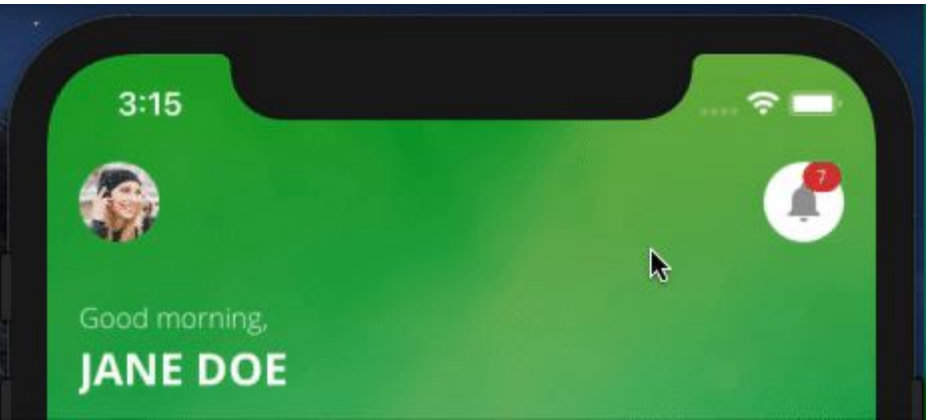
São os componentes do contêiner de navegação nativa ao React Native. Ele não foi projetado para ser usado como uma biblioteca independente, mas sim como uma dependência de uma biblioteca de navegação @react-navigation.

React Navigation - Bibliotecas adicionais

expo install / yarn add / npm i

```
yarn add react-native-screens
yarn add react-native-safe-area-context
yarn add react-native-gesture-handler
yarn add react-native-reanimated
```

Fornece uma API para acessar informações inseridas na área segura do dispositivo. Isso permite que você posicione seu conteúdo adequadamente em torno de entalhes, barras de status, indicadores iniciais e outros elementos de interface de dispositivo e sistema operacional. Ele também fornece um SafeAreaView componente que você pode usar no lugar View para inserir automaticamente suas visualizações para considerar áreas seguras.

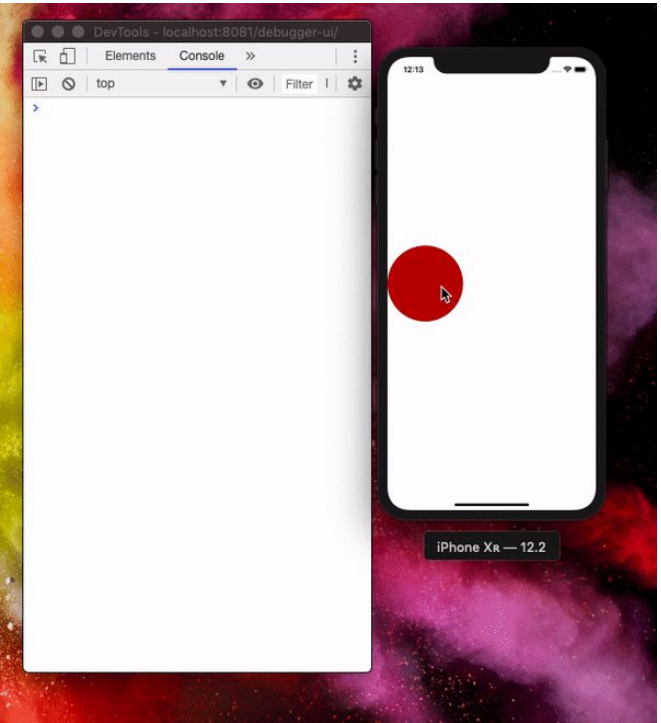


Exemplo de um bug envolvendo o SafeAreaView que já foi corrigido. Mostra o entalhe do celular e como ele funciona.

React Navigation - Bibliotecas adicionais

expo install / yarn add / npm i

```
yarn add react-native-screens
yarn add react-native-safe-area-context
yarn add react-native-gesture-handler
yarn add react-native-reanimated
```



API declarativa expondo o sistema de toque e gesto nativo da plataforma para uso no React Native.

O React Native Gesture Handler fornece APIs de gerenciamento de gestos nativos para criar as melhores experiências baseadas em toques possíveis no React Native.

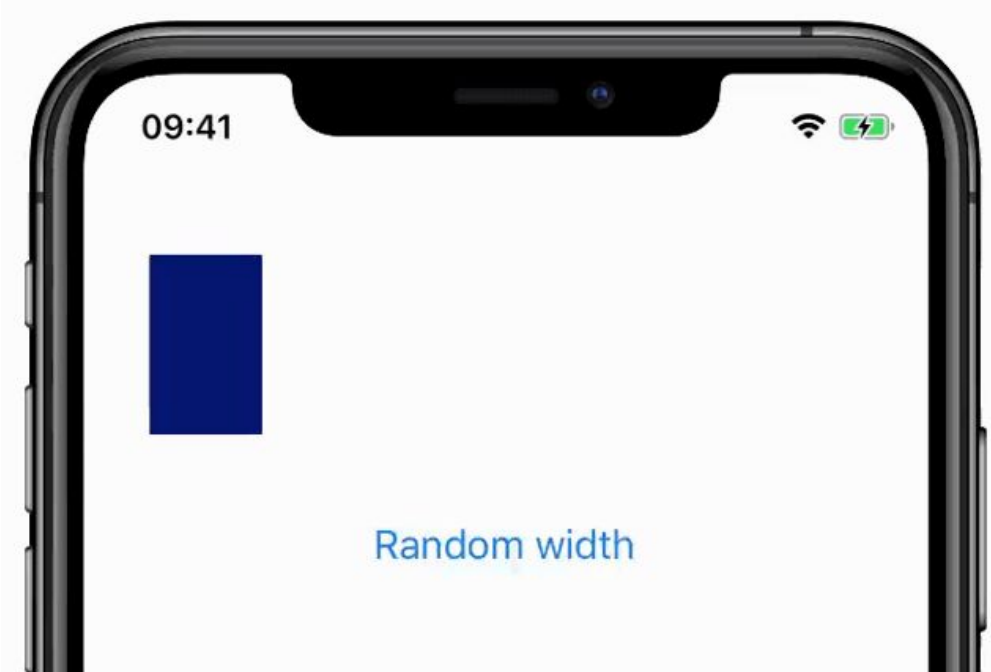
Com essa biblioteca, os gestos não são mais controlados pelo sistema de resposta em Javascript, passam a ser reconhecidos e rastreados no thread da interface do usuário.

Ele torna as interações de toque e o rastreamento de gestos não apenas suaves, mas também confiáveis e seguros.

React Navigation - Bibliotecas adicionais

expo install / yarn add / npm i

```
yarn add react-native-screens
yarn add react-native-safe-area-context
yarn add react-native-gesture-handler
yarn add react-native-reanimated
```



Reanimated é uma biblioteca React Native que permite criar animações e interações suaves que são executadas no thread da interface do usuário.

No React Native, por padrão, todas as atualizações são atrasadas em pelo menos um quadro, pois a comunicação entre a interface do usuário e o thread JavaScript é assíncrona e o thread da interface do usuário nunca espera que o thread JavaScript conclua o processamento de eventos.

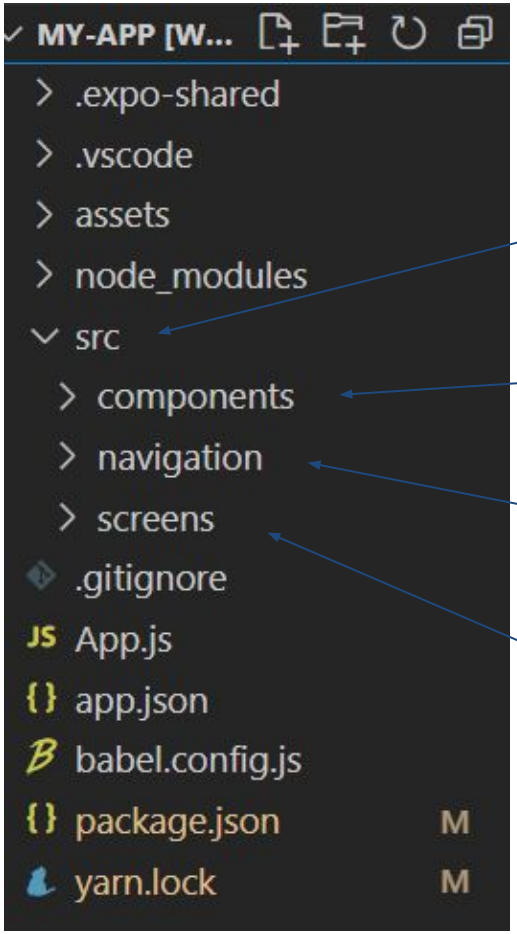
Reanimated tem como objetivo fornecer maneiras de descarregar animações e lógica de manipulação de eventos do thread JavaScript para o thread da interface do usuário de forma mais natural.

React Navigation

React Navigation - Resumo da instalação

```
npx expo init my-app --template blank
cd my-app

yarn add @react-navigation/native @react-navigation/native-stack
@react-navigation/drawer @react-navigation/bottom-tabs
@react-navigation/material-bottom-tabs react-native-paper
react-native-vector-icons react-native-screens react-native-safe-area-context
react-native-gesture-handler react-native-reanimated

code .

expo start:web
```

React Navigation - Estrutura do projeto



src: será onde trabalharemos nosso projeto, contendo suas subpastas organizadas com cada um dos tipos de componentes

src/components: iremos inserir os componentes auxiliares aqui, podendo também conter subpastas para sinalizar cada um.

src/navigation: aqui teremos a navegação do nosso projeto, contendo um arquivo **index.js** que sinaliza a navegação base do nosso app.

src/screens: teremos as telas do nosso aplicativo, cada tela terá seu respectivo nome com nome de componente.

Expo

- O que é Expo e suas funcionalidades básicas: apresentar o Expo como uma plataforma que facilita o desenvolvimento de aplicativos React Native, incluindo recursos como Live Reload, Hot Module Replacement e acesso a APIs nativas.
- Link para documentação do React Native: https://reactnative.dev/docs/getting-started
- Link para documentação do Expo: https://docs.expo.dev/

Node.js

- Configuração do Node.js e do npm: explicar como instalar o Node.js e o npm, que são pré-requisitos para o desenvolvimento de aplicativos React Native.
- Instalação do Expo CLI: apresentar como instalar o Expo CLI, que é a ferramenta de linha de comando do Expo.
- Criação de um novo projeto React Native com Expo: demonstrar como criar um novo projeto React Native usando o Expo CLI.

- Link para documentação do Node.js: https://nodejs.org/en/download/
- Link para documentação do Expo CLI: https://docs.expo.dev/workflow/expo-cli/
- Link para documentação de criação de um novo projeto React Native com Expo: https://docs.expo.dev/get-started/create-a-new-app/

Senac