

Desenvolvimento de { [Aplicativos Móveis]

< Utilizando React Native >

}



```
1  Fabrício Lugão; {  
2  
3
```



```
6  fabriciolugao@gmail.com  
7
```



```
10 https://www.linkedin.com/in/fabriciolugao/  
11  
12  
13  
14
```

```
15 }
```

Formação 'Profissional' {

01 Sistemas de informação

< Faminas 2010 - 2013 >

02 Estágio

< 2º semestre de 2010 >

03 Startup

< Empresa "enxuta";
Desenvolvimento de app;
Desenvolvimento front-end >

}

Formação 'Profissional' {

04 MBA em Projeto de Aplicações
para Dispositivos Móveis

< IGTI 2015 - 2016 >

05 Full-stack / Educação

< Soluções para a área educacional >

06 Trabalho remoto

< 2019 - Prós vs contras >

}

Formação 'Profissional' {

07 Trabalho em uma empresa multinacional: Dell

< 2020 >

< Testes; Inglês >

08 Tech lead

< Design patterns >

< Tecnologias recentes >

< Gerenciamento de pessoas
(feedbacks, motivação) >

< Aplicação do scrum >

}

Formação 'Profissional' {

09 Grupo Sapiens - remoto

[Vantagens]

< Bom convívio com os stakeholders >

< Liberdade de trabalho (horário) >

< Liberdade tecnológica >

[Desafios]

< Gestão do tempo >

< Gestão de projetos >

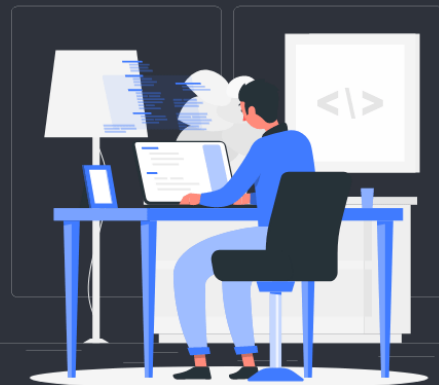
< Se manter atualizado >

}

Desenvolvimento de { [Aplicativos Móveis]

< Utilizando React Native >

}



Google

Android

Aprenda a criar aplicações para dispositivos móveis
com o Android SDK

Ricardo R. J.



9.990
eBook


```
1
2
3
4
5 Uma breve {
6     |
7     |
8     |
9     |
10    |
11    |
12    |
13    |
14 }
```

história do celular;

Symbian OS (C++) {

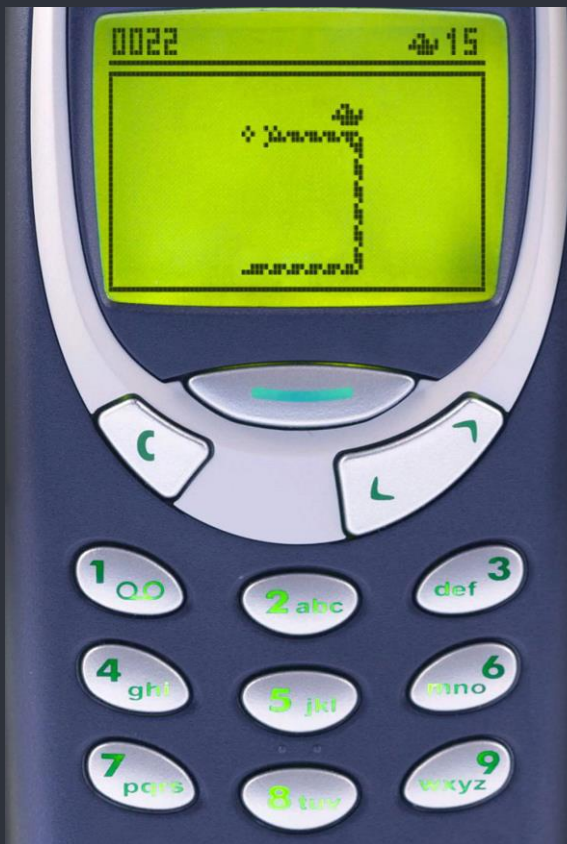
< Funções simples e objetivas >

< Código fechado >

< Poucos menus e apps >

< Última versão em 2012 >

}





Blackberry OS (C++) {

< Suporte para aplicativos em Java >

< Acesso à internet >

< Envio de e-mails >

}

```
Android {  
    | Be together.  
    | Not the same  
}
```



Android {

< 2003 - Symbian (Nokia)
dominava o mercado>

< 2005 - Android Inc.
comprada pelo Google.
(Assim como o YouTube) >

< Competir com a Microsoft
Medo que ela dominasse o
mercado assim como nos PCs >

< Plataforma mobile única e
open source >



1
2
3
4
5
6
7
8
9
10
11
12
13
14

SAMSUNG



LG



motorola



2007

iOS

{

Apple reinvents
the phone.

}

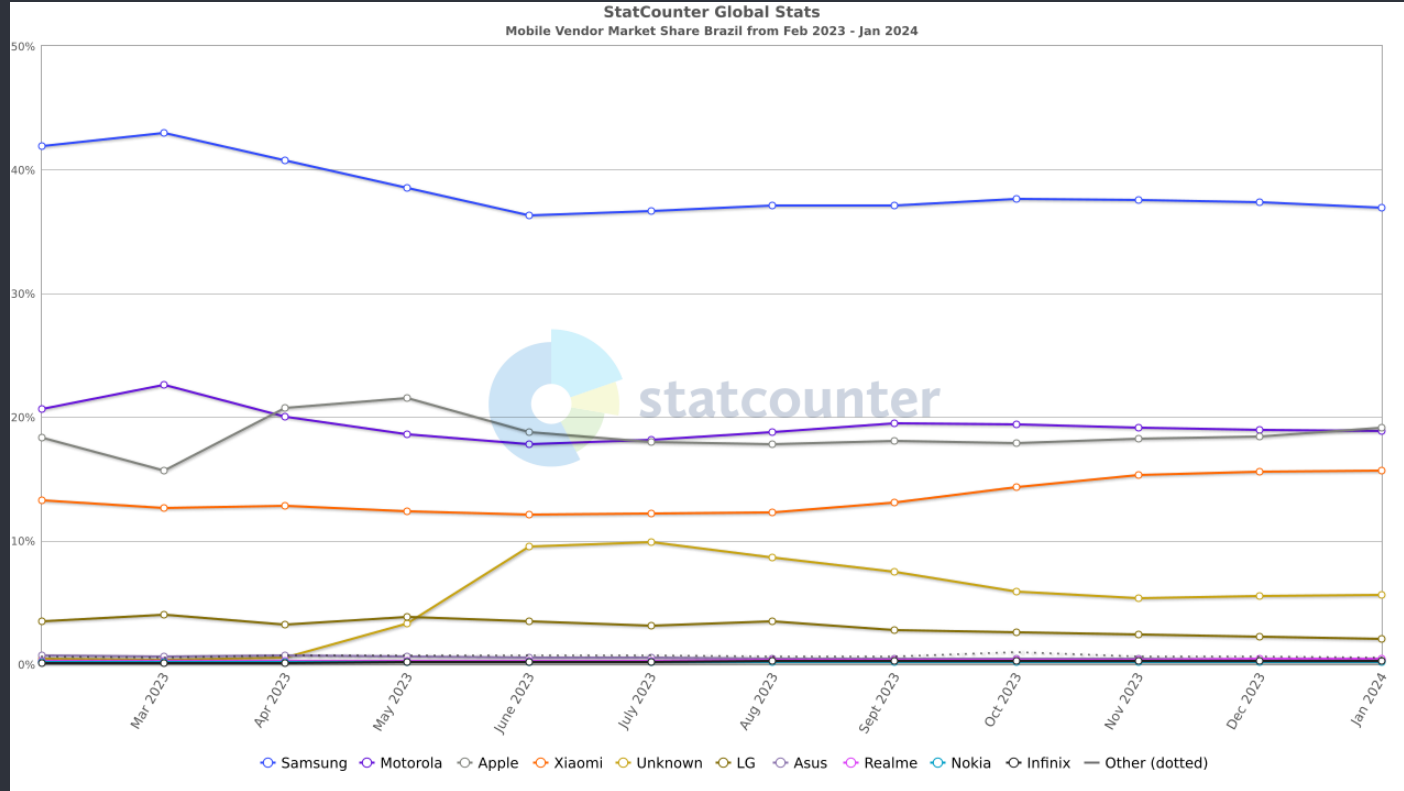



```
1  ios {  
2  
3  
4  
5  
6  
7    < “Definiu” como seria um smartphone >  
8  
9    < Diretrizes de experiência de usuário >  
10  
11    < Código fechado >  
12  
13  }  
14
```



1
2
3
4
5
6
7
8
9
10
11
12
13
14





[Código aberto] vs [Código fechado]



Android {

< É um sistema de **código aberto**, ou seja, os proprietários dos dispositivos possuem a **liberdade de modificar** o sistema usado nos celulares, tablets e afins, o que pode trazer **novas funcionalidades** mas, ao mesmo tempo, **prejudicar a segurança** >

}

iOS {

< É um **sistema fechado**, cujo código-fonte não é compartilhado nem mesmo com os desenvolvedores de aplicativos. Assim, eles devem **seguir às regras da Apple**, o que a coloca no topo da cadeia de **segurança** e permite que potenciais problemas já sejam cortados pela raiz. >

}

Desenvolvimento { de Aplicativos;

|
}



Desenvolvimento Nativo {

< A aplicação mobile que desenvolvemos é específica de uma única plataforma ou sistema, e é feita com as linguagens e ferramentas específicas daquela plataforma >

< android > * Java
 * Kotlin (2027)

< iOS > * Objective-C
 * Swift (2014)

}

IDEs {

< Ambiente de desenvolvimento integrado >



}


```
1
2
3  Distribuição nas 'lojas' {
4
5
6
7
8
9
10
11
12
13
14
```

Distribuição nas 'lojas' {



}

Monetização {



}

Monetização {

< android >

- * Menos aplicativos pagos na loja
- * Usuários são muito mais abertos para propaganda durante o uso

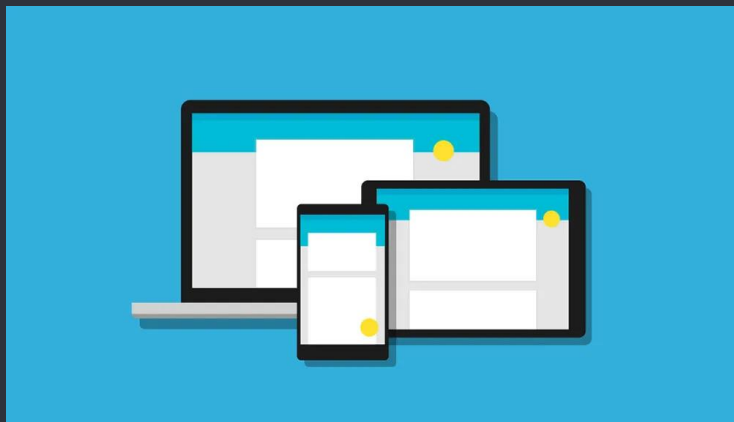
< iOS >

- * Mais apps pagos ou “freemium”
- * Usuários menos abertos a banners e propagandas durante o uso do app

}

Design Guidelines “Diretrizes de layout” {

< Material Design X Human Interface Guideline >



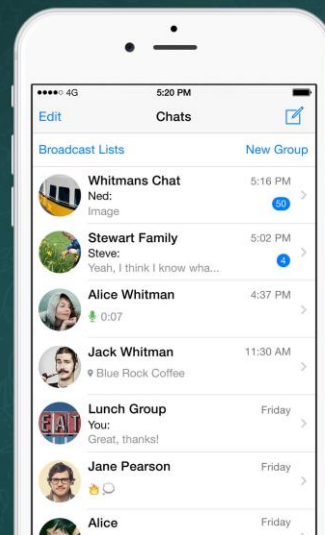
}

Design Guidelines “Diretrizes de layout” {

Simples. Pessoal.
Mensagens em tempo real.



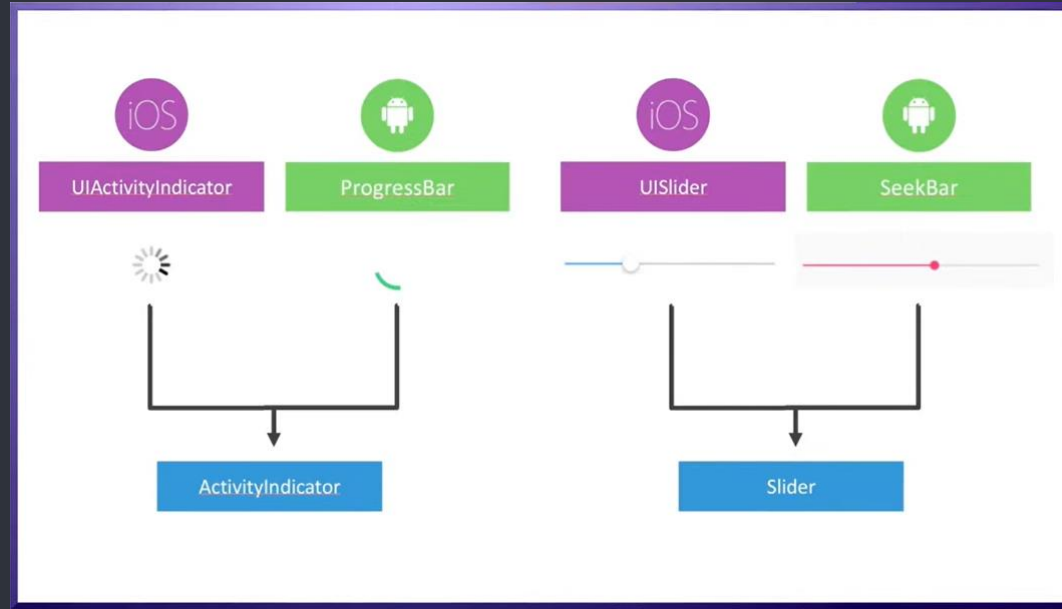
Simple. Personal.
Real time messaging.



Aplicativos 'híbridos';

- * Essa abordagem envolve a criação de um **único aplicativo** que funciona em várias plataformas, como iOS e Android.
- * Os desenvolvedores podem usar de tecnologias da web, como HTML, CSS e JavaScript, para criar o aplicativo, que é então **encapsulado em um contêiner nativo** e executado como um aplicativo nativo

Aplicativos 'híbridos' {



```
1  Aplicativos 'híbridos' vs Aplicativos 'nativos' {
2
3      * Linguagens nativas reduzem as falhas de segurança
4
5      * Possuem melhor experiência para os usuários
6
7      * Quanto ao custo, seu valor é elevado, ainda mais considerando
8        a equipe de desenvolvedores necessária para entregar o
9        aplicativo final.
10
11     * Já o desenvolvimento híbrido é recomendado para não demandem
12       performance robusta e funções avançadas
13       (Câmera, Mapas, GPS)
14 }
```



```
1  Aplicativos 'híbridos' vs Aplicativos 'nativos' {
2
3      * Linguagens nativas reduzem as falhas de segurança
4
5      * Possuem melhor experiência para os usuários
6
7      * Quanto ao custo, seu valor é elevado, ainda mais considerando
8        a equipe de desenvolvedores necessária para entregar o
9        aplicativo final.
10
11     * Já o desenvolvimento híbrido é recomendado para não demandem
12       performance robusta e funções avançadas
13       (Câmera, Mapas, GPS)
14 }
```

Aplicativos 'híbridos' vs Aplicativos 'nativos' {

< startup >



< app de controle de
estoque para uma empresa >



}

Desenvolvimento híbrido {

< ferramentas >

* Flutter (Dart)

* Ionic

* Xamarin -> .NET Multi-Platform
App UI (C#)

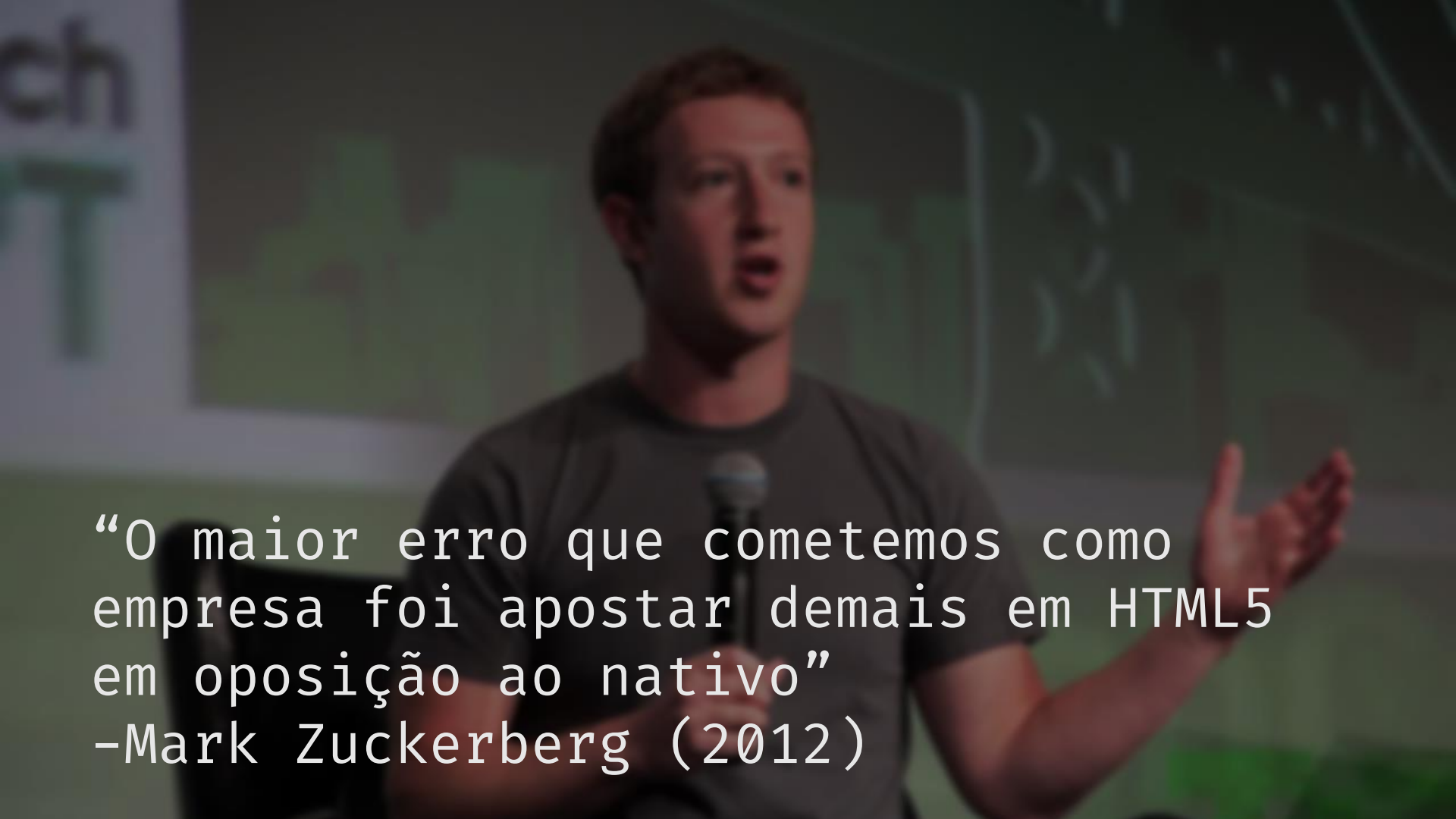
* Cordova

}



React Native

Learn once, write anywhere.

A photograph of Mark Zuckerberg speaking at a conference. He is wearing a dark t-shirt and has his right hand raised in a gesture. The background is a blurred stage with a large screen displaying the word "HTML" in green and white. The text overlay is in a white, monospaced font.

“O maior erro que cometemos como
empresa foi apostar demais em HTML5
em oposição ao nativo”
-Mark Zuckerberg (2012)

React Native;

- * Baseado em react-native.js
- * Possibilita a criação de aplicações móvel multiplataforma (Android e iOS) utilizando apenas **Javascript**
- * Todo o código desenvolvido é convertido para **linguagem nativa** do sistema operacional, o que torna o app muito mais **fluido**

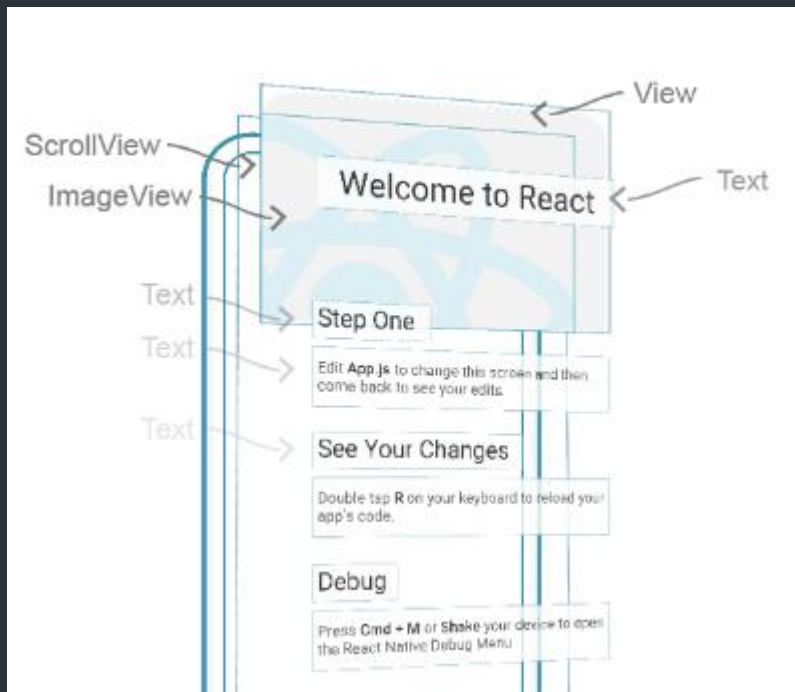
React Native;

- * Baseado em react-native.js
- * Possibilita a criação de aplicações móvel multiplataforma (Android e iOS) utilizando apenas **Javascript**
- * Todo o código desenvolvido é convertido para **linguagem nativa** do sistema operacional, o que torna o app muito mais **fluido**

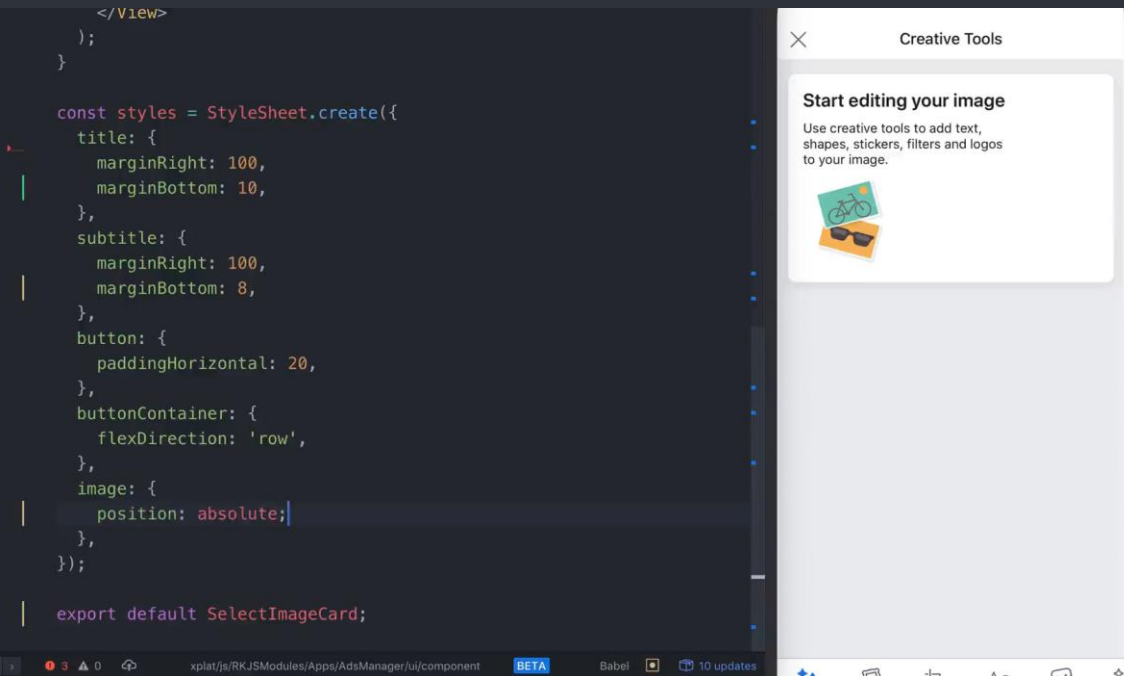
React Native {

- * Fornece um conjunto básico de componentes nativos independentes de plataforma, como View, Text, e Image
- * São mapeados diretamente para os blocos de construção da interface do usuário nativos da plataforma

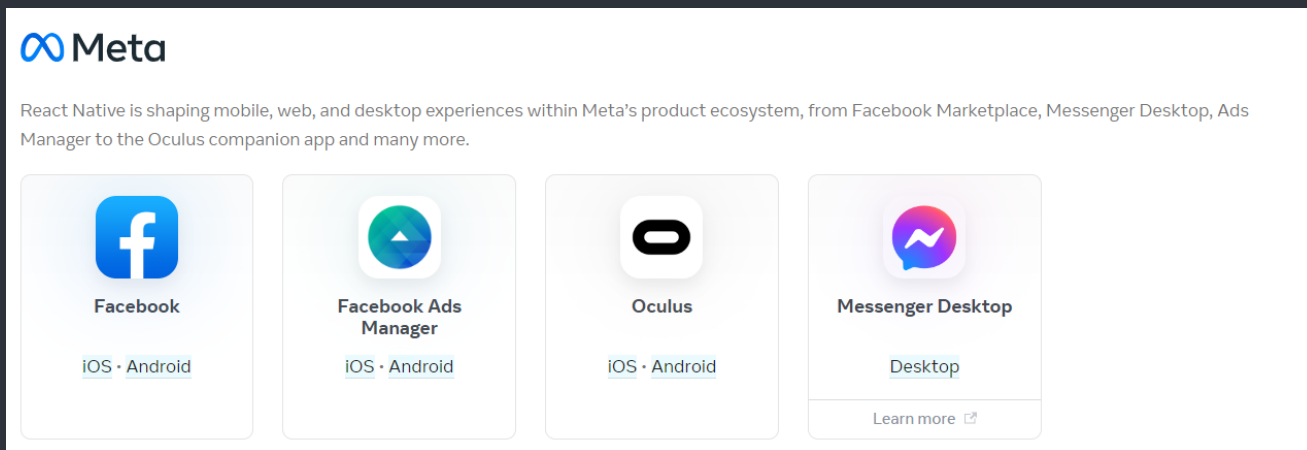
}



React Native | Atualização rápida;

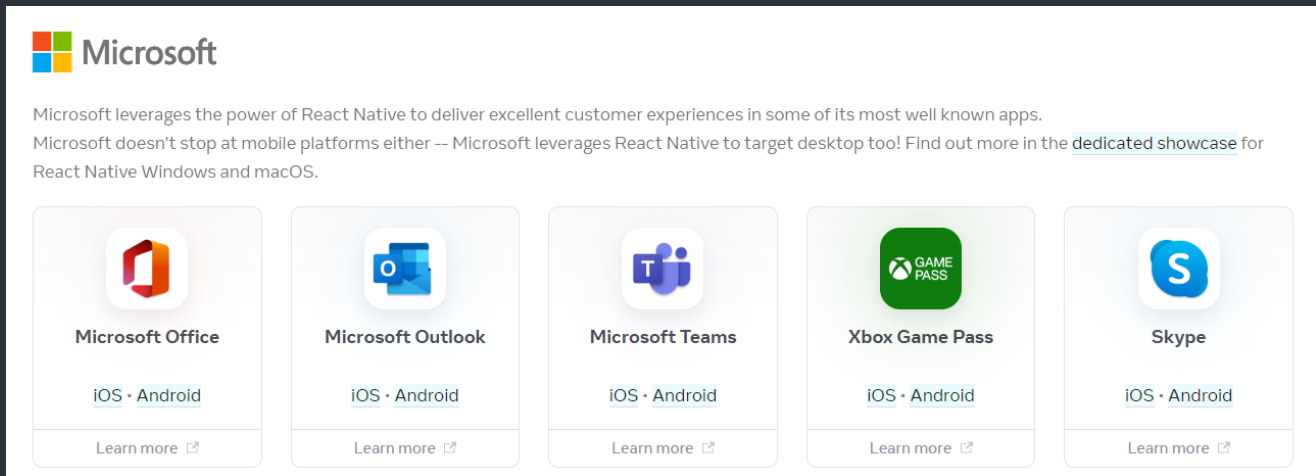


Quem utiliza {








}

Quem utiliza {



Microsoft

Microsoft leverages the power of React Native to deliver excellent customer experiences in some of its most well known apps. Microsoft doesn't stop at mobile platforms either -- Microsoft leverages React Native to target desktop too! Find out more in the [dedicated showcase](#) for React Native Windows and macOS.

 Microsoft Office iOS · Android Learn more	 Microsoft Outlook iOS · Android Learn more	 Microsoft Teams iOS · Android Learn more	 Xbox Game Pass iOS · Android Learn more	 Skype iOS · Android Learn more
---	--	---	---	--

Quem utiliza {



Amazon has used React Native to rapidly deliver new customer-facing features in some of its most popular mobile applications as early as 2016. Amazon also uses React Native to support customer-favorite devices such as the Kindle E-readers.



Amazon Shopping

[iOS](#) • [Android](#)



Amazon Alexa

[iOS](#) • [Android](#)



Amazon Photos

[iOS](#) • [Android](#)



Amazon Kindle

[Learn more](#)



Amazon Appstore

[Learn more](#)

}

Quem utiliza {



All new mobile apps at Shopify are React Native and we are actively migrating our flagship merchant admin app Shopify Mobile to React Native as well. You can read more about React Native development at Shopify on our [blog](#).



Shopify

[iOS](#) · [Android](#)



Shop: All your favorite brands

[iOS](#) · [Android](#)



Shopify Inbox

[iOS](#) · [Android](#)

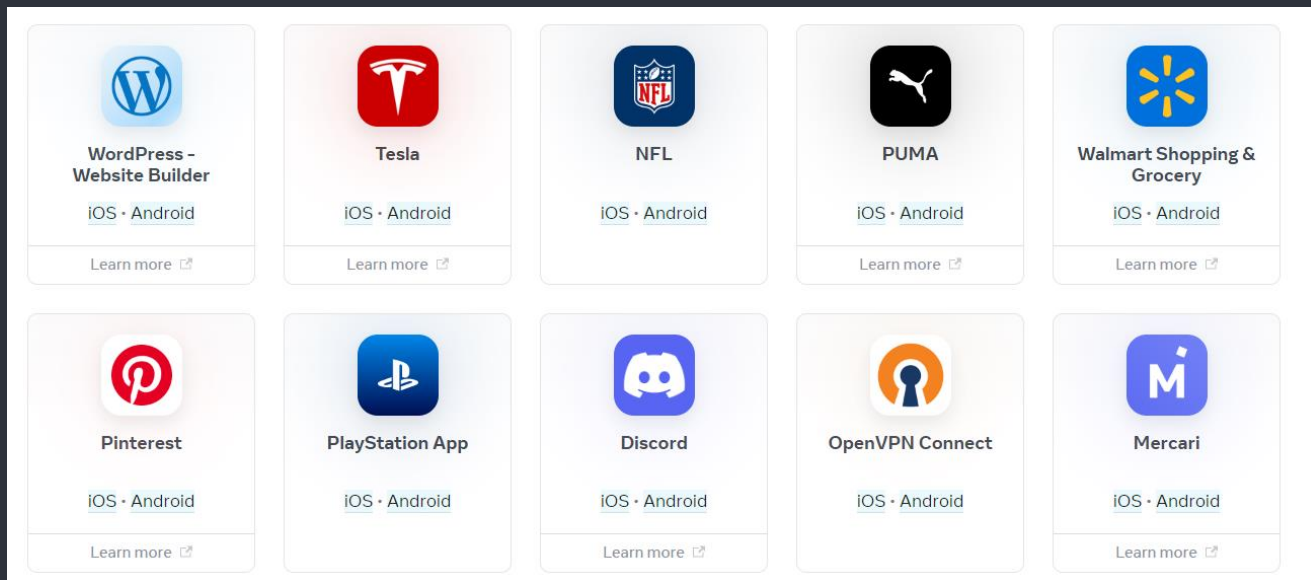


Shopify Point of Sale

[iOS](#) · [Android](#)

}

Quem utiliza {



```
1
2
3
4
5   export default function App() {
6     return (
7       <View>
8         <Text>Hello world!</Text>
9       </View>
10    );
11  }
12
13
14
```


Componentes são compilados {

Navegador (react-dom)	Componente nativo (Android)	Componente nativo (iOS)	React Native JS
<div>	android.View	UIView	<View>
<input>	EditText	UITextField	<TextInput>
...

}

E a lógica? {

1
2
3
4
5
6
7
8
9
10
11
12
13
14

}

Elementos de
interface

Componentes expostos
pelo React Native



lógica

Escrita por nós, em
JavaScript



Thread Javascript,
hospedada pelo React
Native (no app)

Compilados para
componentes nativos

Não é compilada!

< Threads: conjunto de instruções dentro de uma tarefa maior >

Criando um novo app React Native {

Expo CLI



React Native CLI

Serviço de terceiro
(grátis!)

Desenvolvido pelo time React
Native e comunidade

Gerencia o desenvolvimento
do app

Desenvolvimento básico
(você precisa configurar
muito mais)

Muito prático, menos atrito

Menos features práticas

Você pode deixar o
ecossistema Expo a
qualquer momento

Fácil integração com o
código fonte nativo

Expo

Expo é um ecossistema de
ferramentas que ajudam você



desenvolver



análisar &



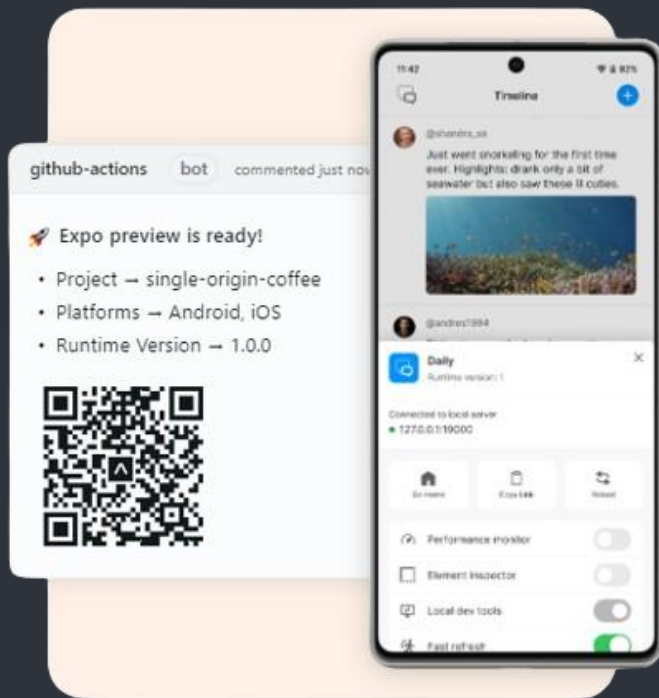
implantar

Crie aplicativos nativos universais com React que rodam em Android, iOS e na web. Itere com confiança.

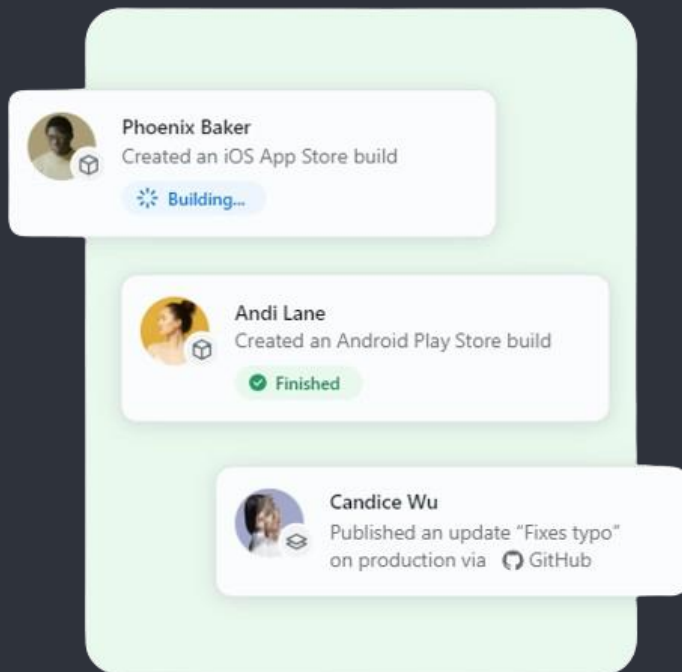
Expo



Expo



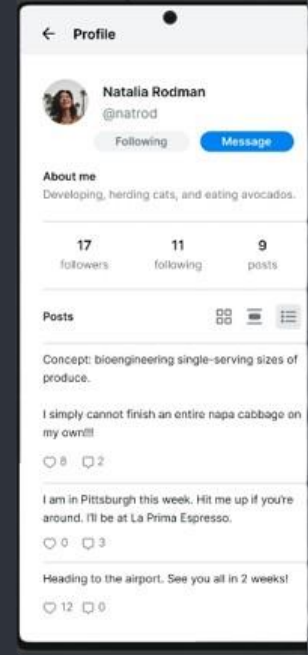
Expo



faminas-2024.html

react-native.js

Expo



Desenvolvimento de Aplicativos Móveis

Expo Go

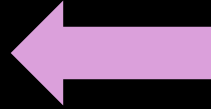
- Expo Go é um sandbox gratuito e de código aberto para aprender e experimentar React Native em dispositivos Android e iOS.
- Você pode instalá-lo diretamente das lojas de aplicativos e colocá-lo em funcionamento em minutos – sem necessidade de instalar um conjunto de ferramentas nativo e compilar um aplicativo.

Expo Go

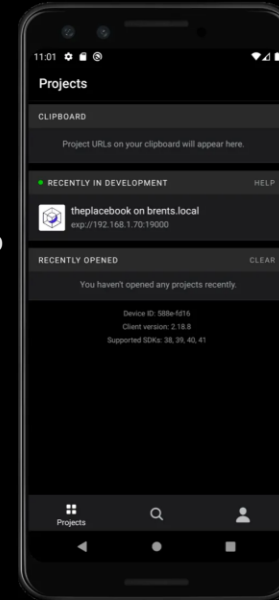
Expo CLI



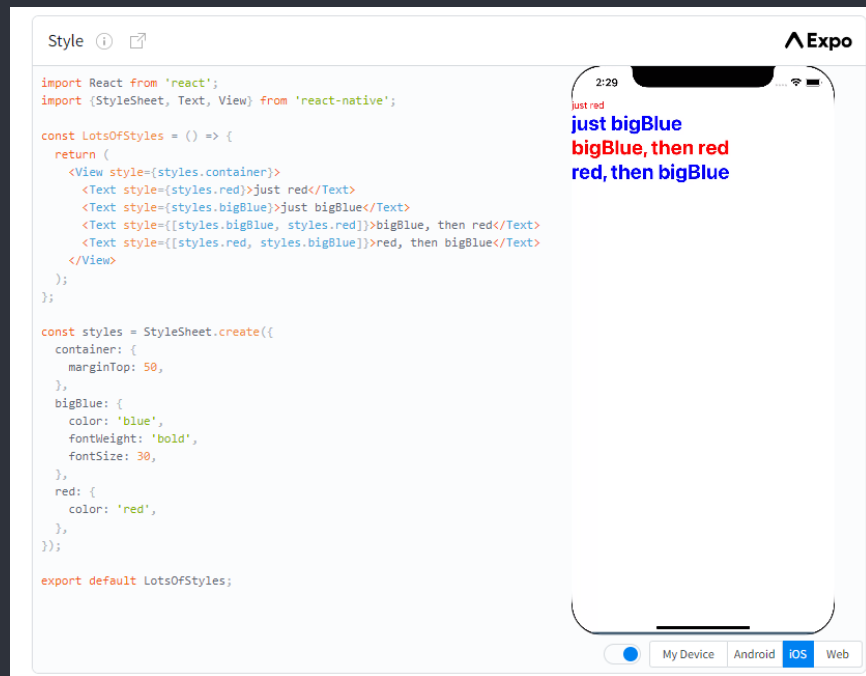
Carrega o projeto



Expo Go



Documentação



Configuração de ambiente

```
npx create-expo-app FaminasADS
```

```
cd FaminasADS
```

```
npx expo start
```

```
}
```

Configuração de ambiente

```
< Parabéns! >
```

```
< Você executou seu primeiro app em React  
Native com sucesso! >
```



Simulador



Android Studio

Principais conceitos

- Utilizar os componentes React Native e construir interfaces;
- Estilizar aplicativos React Native;
- Adicionar interatividade e gerenciar estado;

}

1
2
3
4
5
6
7
8
9
10
11
12
13
14

Lista de tarefas

Não existe CSS!

Estilos inline



Objetos StyleSheet
(folha de estilo)

Escrito em Javascript

Baseado na sintaxe CSS, mas apenas um subconjunto de
propriedades e features são aceitas

Flexbox

- Layouts são tipicamente criados com Flexbox;
- Muito parecido com o Flexbox CSS nos navegadores;
- Define como os elementos ficarão posicionados dentro de containers;
- Posicionamento é controlado via as configurações de estilo aplicadas no elemento container;

}

Flexbox

```
flex: 1,  
flexDirection: 'column',  
justifyContent: 'flex-start',  
alignItems: 'flex-start'
```

Controla as orientações
dos eixos

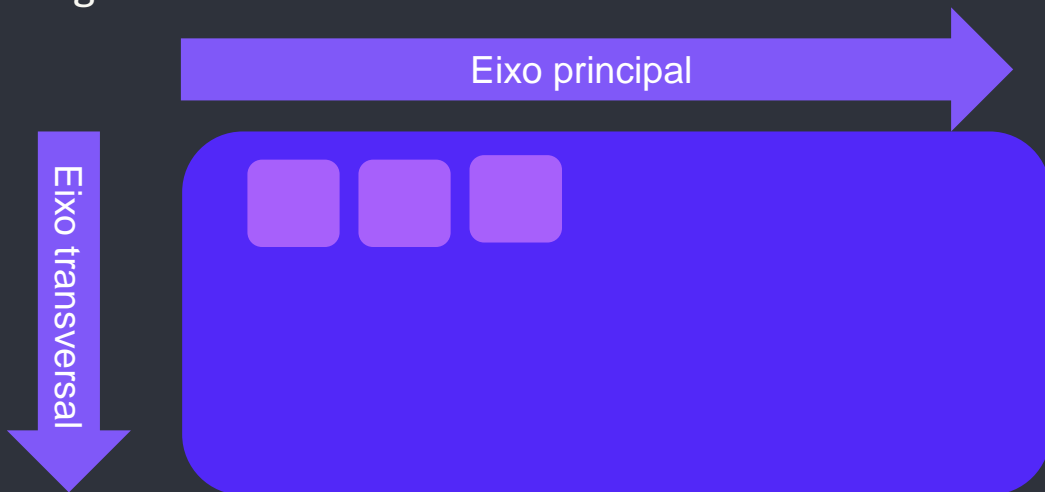
Eixo transversal

Eixo principal



Flexbox

```
flex: 1,  
flexDirection: 'row',  
justifyContent: 'flex-start',  
alignItems: 'flex-start'
```



Flexbox

```
flex: 1,  
flexDirection: 'row',  
justifyContent: 'space-between',  
alignItems: 'flex-start'
```



Flexbox

- Toda View, por padrão utiliza flexbox;
- Toda View, por padrão utiliza flexDirection: 'column'. (De cima para baixo)

}

Flexbox

- Toda View, por padrão utiliza flexbox;
- Toda View, por padrão utiliza flexDirection: 'column'. (De cima para baixo)

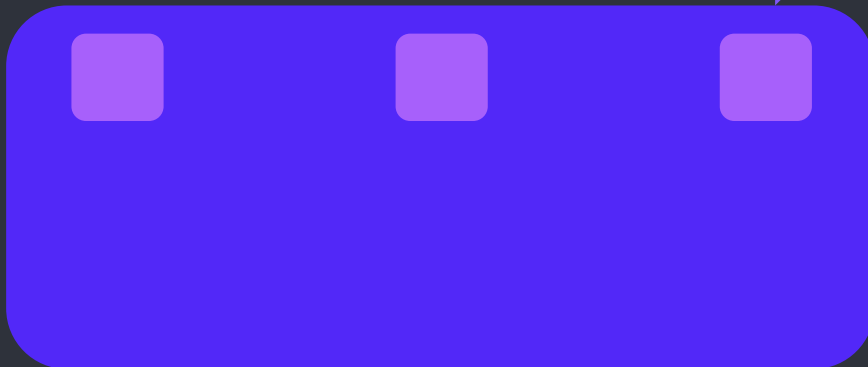
}

Flexbox

```
flex: 1,  
flexDirection: 'row',  
justifyContent: 'space-between',  
alignItems: 'flex-start'
```

Eixo principal

Eixo transversal



1
2
3
4
5
6
7
8
9
10
11
12
13
14

Lista de tarefas

Adicionar espaçamento

```
1
2
3      const styles = StyleSheet.create({
4        container: {
5          flex: 1,
6          paddingTop: 50,
7          paddingHorizontal: 16
8        },
9      });
10
11
12
13 }
14
```

Adicionar estilo input


```
<TextInput style={styles.textInput} placeholder='Digite sua tarefa' />
```

```
textInput: {  
  width: '70%',  
  borderColor: "#ccc",  
  borderWidth: 1,  
  marginRight: 8,  
  padding: 8,  
}
```



```
}
```


Alinha botão



```
1 <View style={styles.inputContainer}>
2   <TextInput style={styles.textInput} placeholder='Digite sua tarefa' />
3   <Button title='Adicionar' />
4 </View>
5
6   inputContainer: {
7     flexDirection: "row",
8     justifyContent: "space-between",
9     alignItems: "center",
10    borderBottomWidth: 1,
11    borderBottomColor: "#ccc"
12  }
13
14
```

Dividir tela

```
1
2
3
4   <View style={styles.container}>
5     <View style={styles.inputContainer}>
6       <TextInput style={styles.textInput} placeholder="Digite sua tarefa" />
7       <Button title="Adicionar" />
8     </View>
9     <View style={styles.tarefasContainer}>
10    </View>
```



Dividir tela

```
1  
2  
3      inputContainer: {  
4          flexDirection: "row",  
5          justifyContent: "space-between",  
6          alignItems: "center",  
7          borderBottomWidth: 1,  
8          borderBottomColor: "#ccc",  
9          flex: 1,  
10     },  
11     tarefasContainer: {  
12         flex: 5,  
13     },  
14
```



Manipulando eventos

- A manipulação de eventos é feita da mesma forma que nos aplicativos web
- Podemos adicionar ouvintes de eventos e conectá-los a funções manipuladoras de eventos
- Podemos gerenciar o estado do seu componente com o Hook `useState`

}

1 Manipulando eventos

2

3

4

5

6

7

8

9

10

11

12

13


14

```
function textoTarefaManipulador() {  
}  
  
function adicionarTarefaManipulador() {  
}
```

```
}
```


Manipulando eventos

```
1  
2  
3  
4  
5  
6      <TextInput  
7        style={styles.textInput}  
8        placeholder="Digite sua tarefa"  
9        onChangeText={textoTarefaManipulador}  
10     />  
11  
12  
13 }  
14
```



Manipulando eventos

```
1  
2  
3  
4  
5  
6     var tarefaDigitada;
```

```
7  
8     function textoTarefaManipulador(textoDigitado) {  
9         tarefaDigitada = textoDigitado;  
10    }
```

```
11  
12  
13 }  
14
```

Manipulando eventos

```
1  
2  
3  
4  
5  
6     var tarefaDigitada;
```

```
7  
8     function textoTarefaManipulador(textoDigitado) {  
9         tarefaDigitada = textoDigitado;  
10    }
```

```
11  
12  
13 }  
14
```

1 Manipulando eventos

2

3

4

5

6

7

8

```
<Button title="Adicionar" onPress={adicionarTarefaManipulador} />
```

9

10

11

12

13 }

14

Gerenciamento de estado

- Antes de seguirmos, vamos aprender um conceito importante sobre o gerenciamento de estado de um app

```
1
2
3
4
5
6
7
8
9
10
11
12
13 }
14
```

Gerenciamento de estado

- Antes de seguirmos, vamos aprender um conceito importante sobre o gerenciamento de estado de um app

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13 }  
14
```

Gerenciamento de estado

```
1  
2  
3  
4  
5  
6  
7  
8   var tarefas = ["Estudar", "Fazer exercícios"];  
9  
10  
11  
12  
13 }  
14
```

Array.map()

- O método `Array.map()` permite a você iterar sobre o array e modificar seus elementos usando uma função de callback. A função de callback será executada em cada um dos elementos do array.

```
let arr = [1, 2, 3, 4];

let arrModificado = arr.map((element) => (
  element * 3;
));

console.log(arrModificado); // [3, 6, 9, 12]
```


Gerenciamento de estado

```
1  
2  
3  
4  
5  
6      <View style={styles.tarefasContainer}>  
7          <Text>{tarefas[0]}</Text>  
8      </View>  
9  
10  
11  
12  
13  }  
14
```

Gerenciamento de estado

```
1  
2  
3  
4  
5  
6      <View style={styles.tarefasContainer}>  
7          <Text>{tarefas[0]}</Text>  
8      </View>  
9  
10  
11  
12  
13  }  
14
```

Gerenciamento de estado

```
1  
2  
3  
4  
5  
6  
7   {tarefas.map((tarefa) => (  
8     <Text>{tarefa}</Text>  
9   ))}  
10  
11  
12  
13 }  
14
```

Gerenciamento de estado

```
1  
2  
3  
4  
5  
6     function adicionarTarefaManipulador() {  
7         tarefas.push(tarefaDigitada);  
8         console.log(tarefas);  
9     }  
10  
11  
12  
13 }  
14
```

React State

- React não renderiza novamente a tela quando um evento é disparado em qualquer momento no aplicativo
- Devemos avisar ao React quando ele deve renderizar novamente (atualizar) utilizando um conceito chamado “state”

}

useState Hook

- Um Hook é um tipo de função que permite “conectar-se” (hook into) aos recursos do React. Por exemplo, useState é um hook que permite adicionar estado aos componentes da função

```
const [tarefas, setTarefas] = useState(["Estudar", "Fazer exercícios"]);
```

```
}
```

useState Hook

```
const [tarefas, setTarefas] = useState(["Estudar", "Fazer exercícios"]);
```

useState faz duas coisas:

- Cria uma “variável de estado” com um valor inicial. (Nesse caso o valor inicial é um array de strings)
- Cria uma função para setar o valor na variável de estado (setTarefas)

}

Gerenciamento de estado


```
1  
2  
3  
4  
5  
6     function adicionarTarefaManipulador() {  
7         setTarefas((tarefasAtuais) => [...tarefasAtuais, tarefaDigitada]);  
8     }  
9  
10  
11  
12  
13 }  
14
```


Gerenciamento de estado

```
1  
2  
3  
4  
5  
6 function adicionarTarefaManipulador() {  
7   console.log(tarefaDigitada);  
8   setTarefas((tarefasAtuais) => [...tarefasAtuais, tarefaDigitada]);  
9 }  
10  
11  
12  
13 }  
14
```

Gerenciamento de estado

```
1  
2  
3  
4     const [tarefaDigitada, setTarefaDigitada] = useState("");  
5  
6     function textoTarefaManipulador(textoDigitado) {  
7         setTarefaDigitada(textoDigitado);  
8     }  
9  
10    <TextInput  
11        style={styles.textInput}  
12        placeholder="Digite sua tarefa"  
13        onChangeText={textoTarefaManipulador}  
14        value={tarefaDigitada}  
15    />
```



Alterações de estilo

```
1 <View style={styles.tarefasContainer}>
2
3   {tarefas.map((tarefa) => (
4     <Text style={styles.tarefa}>{tarefa}</Text>
5   ))}
6 </View>
```

```
7
8   tarefa: {
9     margin: 8,
10    borderRadius: 6,
11    backgroundColor: "#5e0acc",
12    color: "white",
13    padding: 8,
14  },
```

```
}
```

ScrollView



```
<ScrollView style={styles.tarefasContainer}>  
  {tarefas.map((tarefa) => (  
    <Text>{tarefa}</Text>  
  ))}  
</ScrollView>
```

```
}
```

ScrollView



```
<View style={styles.tarefasContainer}>
  <ScrollView>
    {tarefas.map((tarefa) => (
      <Text>{tarefa}</Text>
    ))}
  </ScrollView>
</View>
```

```
}
```


ScrollView

ScrollView renderiza todo o conteúdo que está dentro dela, mesmo que ele não esteja visível para o usuário.

Imagine o cenário com 100, 1000 itens. Isso pode causar um problema de desempenho do seu app.

ScrollView é ideal para listas que possuam um conteúdo delimitado, mas não para listas com conteúdo dinâmico

}

FlatList

O FlatList é um componente **facilita a renderização de listas longas e infinitas de itens**. Ele é altamente otimizada para melhor desempenho e oferece uma maneira eficiente de exibir dados em uma lista rolável, o que a torna ideal para aplicativos que precisam mostrar muitos itens de maneira eficaz

A principal vantagem da FlatList em comparação com outros métodos de exibição de listas, como o ScrollView, é sua eficiência na renderização de grandes conjuntos de dados. **Ele só renderiza os itens visíveis na tela, reciclando componentes conforme o usuário rola a lista, economizando assim recursos de CPU e memória.**

FlatList atributos

data: espera o dados da lista

renderItem: espera uma função como valor que dirá a lista como renderizar os itens individualmente. Esta função esperar um parâmetro que contém o valor do item mais alguns metadados

}

FlatList atributos

```
1 <FlatList
2
3
4   data={tarefas}
5   renderItem={({itemData}) => {
6     return (
7       <Text style={styles.itemTarefa} key={itemData.item}>
8         {itemData.item}
9       </Text>
10     )
11   }}
12 />
13 }
14
```

Dividindo nossos componentes

É uma boa prática quebrar nossos componentes em componentes menores.

Essa é uma prática recomendada para manter nosso código sustentável.



Dividindo nossos componentes

```
1  
2  
3  
4  
5  
6     function TarefaItem() {};  
7  
8  
9     export default TarefaItem;  
10  
11  
12  
13 }  
14
```

Dividindo nossos componentes

```
1
2
3
4 function TarefaItem() {
5   return (
6     <Text style={styles.itemTarefa} key={itemData.item}>
7       {itemData.item}
8     </Text>
9   )
10 };
11
12 export default TarefaItem;
```

Dividindo nossos componentes


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

```
const styles = StyleSheet.create({});
```

Dividindo nossos componentes

```
import TarefaItem from './components/TarefaItem';
```

```
    <FlatList  
      data={tarefas}  
      renderItem={({itemData}) => {  
        return <TarefaItem />  
      }}  
    />
```



Props

Para passar dados para nossos componentes utilizaremos as props.

Nós podemos esperar qualquer dados, pois somos nós que estamos criando o componente.

Props

```
1  
2  
3  
4  
5 function TarefaItem(props) {  
6     return (  
7         <Text style={styles.itemTarefa} key={itemData.item}>  
8             {props.texto}  
9         </Text>  
10    )  
11 };  
12  
13  
14
```



Props

```
1  
2  
3  
4  
5  
6  
7  
8 return <TarefaItem texto={itemData.item} />  
9  
10  
11  
12  
13  
14
```

Dividindo nossos componentes

```
1  function TarefaInput() {  
2    return (  
3      <View style={styles.inputContainer}>  
4        <TextInput  
5          style={styles.input}  
6          placeholder="Digite sua tarefa"  
7          onChangeText={textoTarefaManipulador}  
8          value={tarefaDigitada}  
9        />  
10       <Button title="Adicionar" onPress={adicionarTarefaManipulador} />  
11     </View>  
12   )  
13 }  
14 export default TarefaInput;
```

Dividindo nossos componentes

```
1  
2  
3  
4  
5  
6  
7  
8 const styles = StyleSheet.create({})  
9  
10  
11  
12  
13  
14
```

Dividindo nossos componentes

Dentro do nosso novo componente `TarefaInput` estamos recebendo o texto digitado pelo usuário e atualizando o estado das tarefas com a ajuda do botão “Adicionar”

Mas o estado está sendo gerenciado no componente `App.js` e o texto digitado do usuário no componente `TarefaInput`


Nós podemos “conversar” com o componente pai passando funções de manipulação de eventos através das props

Dividindo nossos componentes

```
1  function TarefaInput(props) {
2    return (
3      <View style={styles.inputContainer}>
4        <TextInput
5          style={styles.input}
6          placeholder="Digite sua tarefa"
7          onChangeText={textoTarefaManipulador}
8          value={tarefaDigitada}
9        />
10       <Button title="Adicionar" onPress={props.onAdicionarTarefa}
11     />
12     </View>
13   )
14 }
```



Dividindo nossos componentes

```
1
2
3
4
5
6   ...
7   <View style={styles.container}>
8      <TarefaInput onAdicionarTarefa={adicionarTarefaManipulador} />
9     <View style={styles.tarefasContainer}>
10  ...
11
12
13
14
```

Dividindo nossos componentes

```
1
2
3
4 function TarefaInput(props) {
5     const [tarefaDigitada, setTarefaDigitada] = useState("");
6
7     function textoTarefaManipulador(textoDigitado) {
8         setTarefaDigitada(textoDigitado);
9     }
10
11     ...
12
13
14
```


Dividindo nossos componentes



```
function adicionarTarefaManipulador() {  
  setTarefas((tarefasAtuais) => [...tarefasAtuais, tarefaDigitada]);  
}
```



Vamos precisar passar por parâmetro

Dividindo nossos componentes



```
function adicionarTarefaManipulador(tarefaDigitada) {  
  setTarefas(((tarefasAtuais) => [...tarefasAtuais, tarefaDigitada]));  
}
```

Dividindo nossos componentes

TarefaInput.js

```
function adicionarTarefa() {  
  props.onAdicionarTarefa(tarefaDigitada);  
  setTarefaDigitada("");  
}
```

```
<Button title="Adicionar" onPress={adicionarTarefa} />
```

Removendo tarefas

Vamos utilizar o componente **Pressable** para tornar seu conteúdo “pressionável”

Removendo tarefas

```
1
2
3
4
5
6   <Pressable onPress={} >
7     <Text style={styles.itemTarefa} key={props.texto}>
8       {props.texto}
9     </Text>
10  </Pressable>
11
12
13
14
```

Removendo tarefas

App.js

```
function deletarTarefa() {}
```

```
<TarefaItem
```

```
  texto={itemData.item}
```

```
  onDeleteTarefa={deletarTarefa}
```

```
/>
```

Removendo tarefas

TarefaItem.js



```
<Pressable onPress={props.onDeleteTarefa}>  
  <Text style={styles.itemTarefa} key={props.texto}>  
    {props.texto}  
  </Text>  
</Pressable>
```

Removendo tarefas

App.js



```
function deletarTarefa(tarefaDeletada) {  
  setTarefas((tarefasAtuais) => {  
    return tarefasAtuais.filter((tarefa) => tarefa !== tarefaDeletada);  
  });  
}
```


Removendo tarefas

TarefaItem.js

```
function deletarTarefa() {  
  props.onDeleteTarefa(props.texto);  
}
```



```
<Pressable onPress={deletarTarefa}>
```

Removendo tarefas

TarefaItem.js

<Pressable

onPress={deletarTarefa}

android_ripple={{ color: "#DDD" }}

>

Removendo tarefas

```
1
2
3  ➡ <View style={styles.itemTarefa}>
4      <Pressable
5          onPress={deletarTarefa}
6          android_ripple={{ color: "#DDD" }}
7      >
8      ➡ <Text style={styles.itemTarefaTexto} key={props.texto}>
9          {props.texto}
10      </Text>
11      </Pressable>
12  </View>
13
14
```

Removendo tarefas

```
1  
2  
3  
4      itemTarefa: {  
5          backgroundColor: "#5e0acc",  
6          marginBottom: 12,  
7          borderRadius: 6,  
8      },  
9      itemTarefaTexto: {  
10         color: "#FFF",  
11         padding: 8,  
12     },  
13  
14
```

Próximas aulas

- Debugando aplicativos em React Native
- Construir um app para restaurante



Erros acontecem...



Depurando apps React Native

- Entender e lidar com erros
- Utilizar DevTools

Depurando apps React Native

PROBLEMS

3

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

ERROR

ReferenceError: Property 'adicionarTarefa1' doesn't exist

This error is located at:

in TarefaInput (at App.js:29)

in RCTView (at View.js:116)

in View (at App.js:28)

in App (at withDevTools.ios.js:25)

in withDevTools(App) (at renderApplication.js:57)

in RCTView (at View.js:116)

in View (at AppContainer.js:127)

in RCTView (at View.js:116)

in View (at AppContainer.js:155)

in AppContainer (at renderApplication.js:50)

in main(RootComponent) (at renderApplication.js:67), js engine: hermes

□

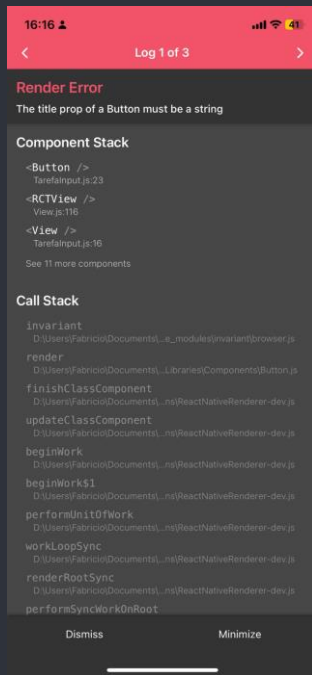
Depurando apps React Native

ERROR Invariant Violation: The title prop of a Button must be a string

This error is located at:

- in Button (at TarefaInput.js:23)
- in RCTView (at View.js:116)
- in View (at TarefaInput.js:16)
- in TarefaInput (at App.js:29)
- in RCTView (at View.js:116)
- in View (at App.js:28)
- in App (at withDevTools.ios.js:25)
- in withDevTools(App) (at renderApplication.js:57)
- in RCTView (at View.js:116)
- in View (at AppContainer.js:127)
- in RCTView (at View.js:116)
- in View (at AppContainer.js:155)

Depurando apps React Native



Depurando apps React Native

Support Ukraine UA [Help Provide Humanitarian Aid to Ukraine.](#)

React Native 0.73 ▾ Development ▾ Contributing Community Showcase Blog 🔍 Search

Core Components ▾

- Core Components and APIs
- ActivityIndicator
- Button
- FlatList
- Image
- ImageBackground
- KeyboardAvoidingView
- Modal
- Pressable
- RefreshControl
- ScrollView
- SectionList
- StatusBar
- Switch
- Text
- TextInput
- TouchableHighlight
- TouchableOpacity
- TouchableWithoutFeedback
- View

Button

A basic button component that should render nicely on any platform. Supports a minimal level of customization.

If this button doesn't look right for your app, you can build your own button using [Pressable](#). For inspiration, look at the [source code](#) for the Button component.

```
<Button
  onPress={onPressLearnMore}
  title="Learn More"
  color="#841584"
  accessibilityLabel="Learn more about this purple button"
/>
```

Example

Button Example ⓘ

```
import React from 'react';
import {
  StyleSheet,
```

The title and onPress handler are required. It is recommended to use

Example

Props


- onPress
- title
- accessibilityLabel
- accessibilityLanguage
- accessibilityActions
- onAccessibilityAction
- color
- disabled
- hasTVPreferredFocus
- nextFocusDown
- nextFocusForward
- nextFocusLeft
- nextFocusRight
- nextFocusUp
- testID
- touchSoundDisabled

Depurando apps React Native

```
console.log('Componente TarefaInput foi renderizado')
```

- Nos auxilia a entender o fluxo do nosso aplicativo
- Adicionando o log em diferentes funções, componentes entendemos qual código está sendo executado em qual momento
- Podemos também passar para o log variáveis para imprimir seus valores

Depurando apps React Native



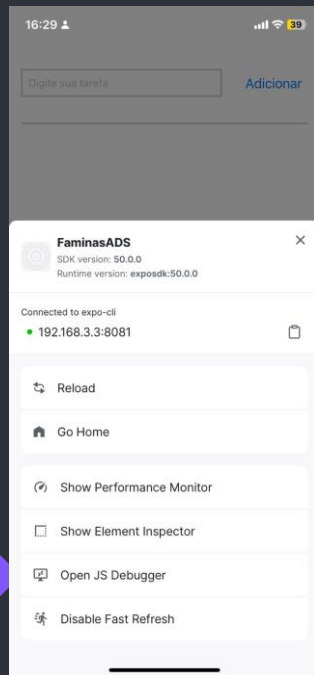
```
> Using Expo Go
> Press s | switch to development build

> Press a | open Android
> Press w | open web

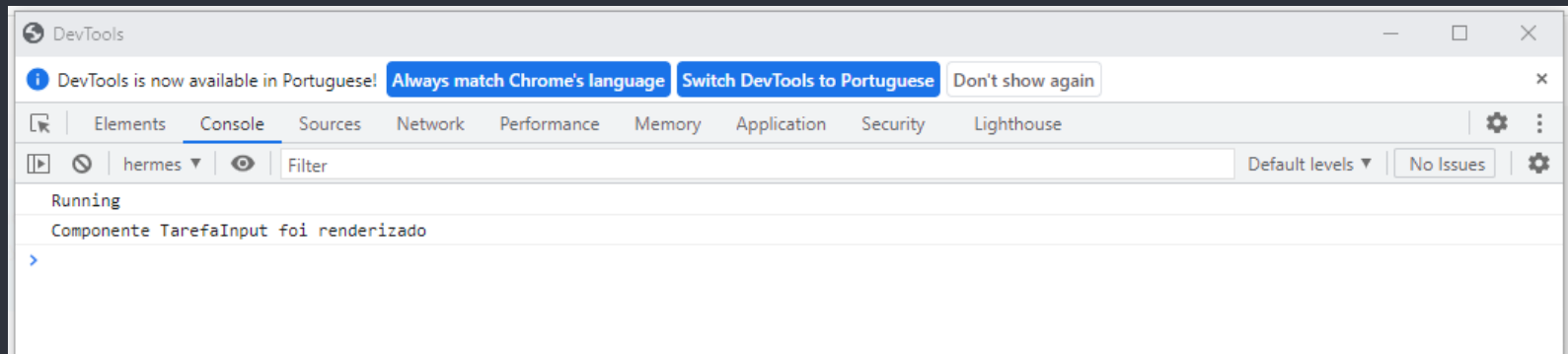
> Press j | open debugger
> Press r | reload app
> Press m | toggle menu
> Press o | open project code in your editor

> Press ? | show all commands
```

Depurando apps React Native



Depurando apps React Native



Utilizando React DevTools

- React DevTools não é uma ferramenta específica para React Native, mas sim a mesma ferramenta utilizada para React para web
- A extensão para navegador não irá funcionar para React Native, temos que instalar no terminal

Utilizando React DevTools

```
npm install -g react-devtools
```

1 Utilizando React DevTools

2

3

4

5

6

7

8

`react-devtools`

9

10

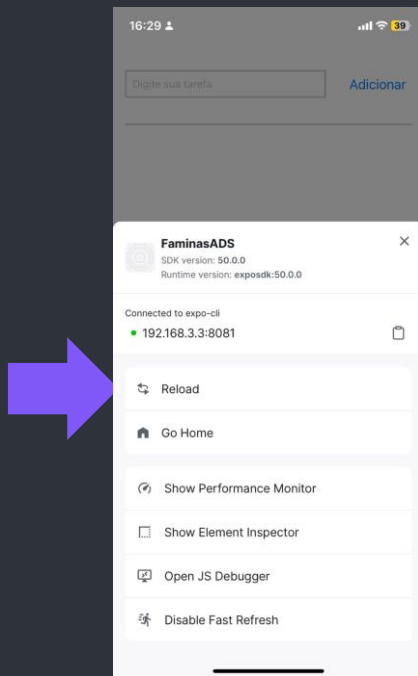
11

12

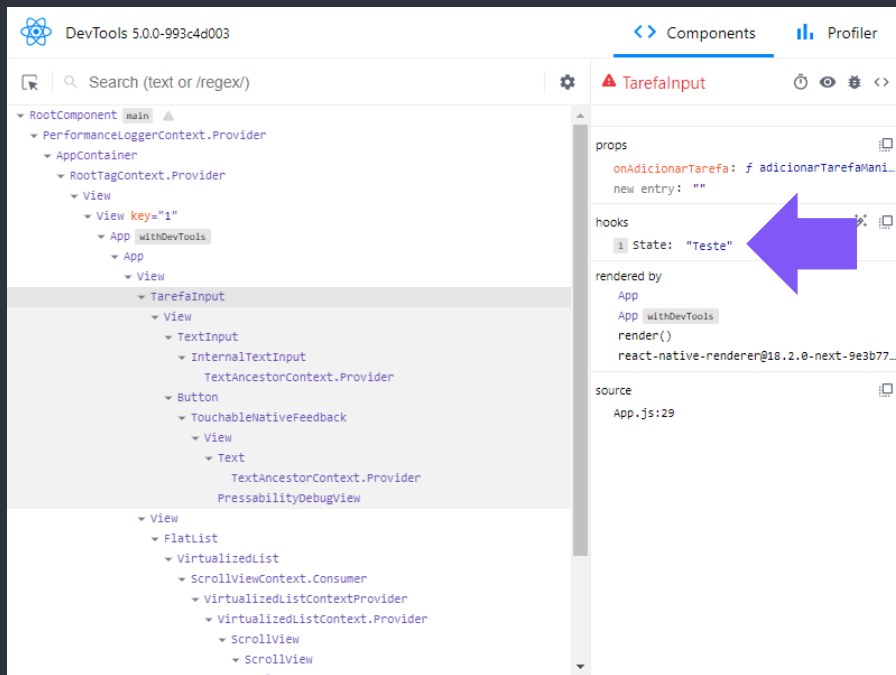
13

14

Utilizando React DevTools

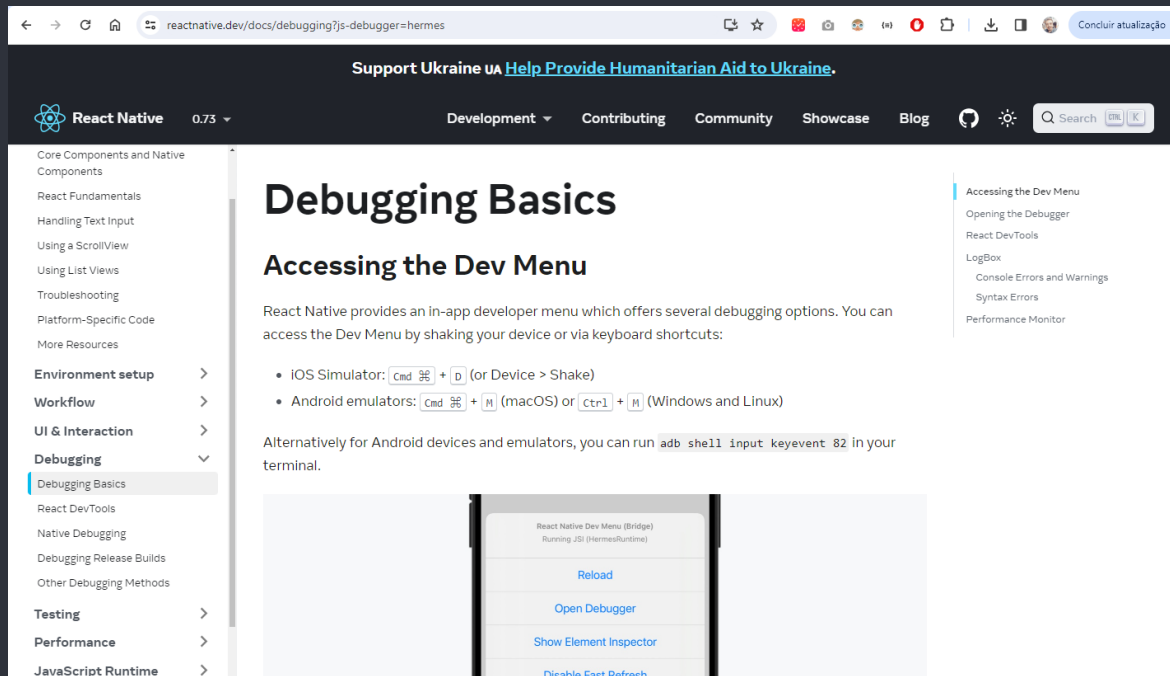


Utilizando React DevTools



Conseguimos
visualizar o
state do
componente e
editá-lo

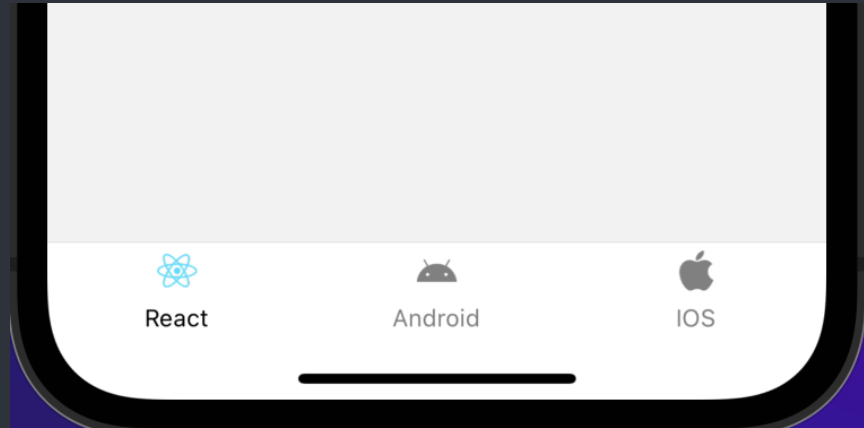
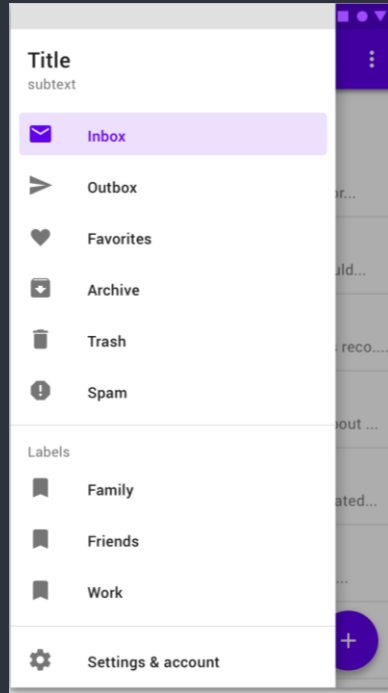
Depurando apps React Native



Navegação

- . Navegação entre telas
- . Utilizando Stack Navigation
- . Drawer e abas

Navegação



Navegação

- Na web utilizamos uma URL para acessar uma página
- E podemos utilizar links e formulários para navegar entre as páginas
- Em apps mobile não temos URLs, navegamos utilizando botões para ir de uma página para outra e também retornar

1 Navegação

2

3

4

5

```
npx create-expo-app Cardapio
```

6

7

8

```
cd Cardapio
```

9

```
npx expo start
```

10

11







12

13

14

```
}
```

Cardápio

1	
2	
3	
4	
5	
6	▼  models 
7	
8	 categoria.js 
9	 prato.js 
10	
11	
12	
13	}
14	

Cardápio

```
1  
2  
3  
4     class Categoria {  
5         constructor(id, titulo, cor) {  
6             this.id = id;  
7             this.titulo = titulo;  
8             this.cor = cor  
9         }  
10    }  
11  
12    export default Categoria;  
13  
14 }
```

Cardápio

```
1
2   class Prato {
3       constructor(
4           id,
5           categoriaIds,
6           titulo,
7           acessibilidadePreco,
8           urlImagem,
9           ingredientes,
10          passos,
11          eVegano,
12          eLivreDeLactose
13      ) {
14          this.id = id,
15          ...
16      }
17  }
18  export default Prato;
```

Cardápio

1
2
3
4
5
6
7
8
9
10
11
12
13
14

}

data
mock-data.js

Cardápio

mock-data.js

```
import Categoria from "../models/categoria";
import Prato from "../models/prato";

export const CATEGORIAS = [
  new Categoria('c1', 'Italiana', '#f5428d'),
  new Categoria('c2', 'Rápido e fácil', '#f54242'),
  new Categoria('c3', 'Lanches', '#f5a442'),
  new Categoria('c4', 'Japonesa', '#f5d142'),
  new Categoria('c5', 'Brasileira', '#368dff'),
  new Categoria('c6', 'Saudável', '#41d95d'),
  new Categoria('c7', 'Sorvetes', '#9eecff'),
  new Categoria('c8', 'Carnes', '#b9ffb0'),
  new Categoria('c9', 'Açaí', '#ffc7ff'),
  new Categoria('c10', 'Peixes', '#47fced')
]
```

Cardápio

mock-data.js

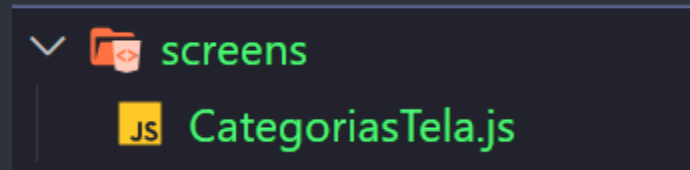
```
1  export const PRATOS = [  
2    new Prato(  
3      'm1',  
4      ['c1', 'c2'],  
5      'Espaguete com molho de tomate',  
6      'barato',  
7      'https://upload.wikimedia.org/wikipedia/commons/thumb/2/20/Spaghetti_Bolognese_mit_Par  
8      _oder_Grana_Padano.jpg/800px-Spaghetti_Bolognese_mit_Parmesan_oder_Grana_Padano.jpg',  
9      [  
10         '4 Tomates',  
11         '1 Colher de chá de azeite',  
12         '1 Cebola',  
13         '250g Espaguete',  
14         'Especiarias e ervas',  
         'Queijo (opcional)'  
      ],  
    ),  
  ],
```

Cardápio

```
1
2   mock-data.js
3   [
4       'Corte os tomates e a cebola em pedaços pequenos.',
5       'Ferva um pouco de água - adicione sal quando ferver.',
6       'Coloque o espaguete na água fervente - eles devem estar prontos em cerca de 10
a 12 minutos.',
7       'Enquanto isso, aqueça um pouco de azeite e adicione a cebola cortada.',
8       'Após 2 minutos, adicione os pedaços de tomate, sal, pimenta e suas outras
especiarias.',
9       'O molho será feito assim que o espaguete estive no ponto.',
10      'Sinta-se à vontade para adicionar um pouco de queijo por cima do prato
acabado.'
11      ],
12      true,
13      false
14  }
```


Cardápio

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13 }  
14
```



Cardápio

CategoriasTela.js

```
import { CATEGORIAS } from "../data/mock-data";
```

```
function CategoriasTela() {  
  return  
}
```

```
export default CategoriasTela
```

```
}
```

Cardápio

CategoriasTela.js

```
function renderItemCategoria(item) {  
  return ;  
}  
  
function CategoriasTela() {  
  return (  
    <FlatList  
      data={CATEGORIAS}  
      keyExtractor={(item) => item.id}  
      renderItem={renderItemCategoria}  
    />  
  );  
}
```

Cardápio

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13 }  
14
```

```
▼ components  
  CategoriaGrid.js
```

Cardápio

CategoriaGrid.js

```
import { Pressable, Text, View } from "react-native";
```

```
function CategoriaGrid({ titulo, cor }) {
```

```
  return (<View>
```

```
    <Pressable>
```

```
      <View>
```

```
        <Text>{titulo}</Text>
```

```
      </View>
```

```
    </Pressable>
```

```
  </View>);
```

```
}
```

```
export default CategoriaGrid;
```

Cardápio

CategoriasTela.js

```
function renderItemCategoria(itemData) {  
  return (  
    <CategoriaGrid titulo={itemData.item.titulo} cor={itemData.item.cor} />  
  );  
}
```

Cardápio


App.js

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

Cardápio

CategoriasTela.js

```
function CategoriasTela() {  
  return (  
    <FlatList  
      data={CATEGORIAS}  
      keyExtractor={({item}) => item.id}  
      renderItem={renderItemCategoria}  
      numColumns={2}  
    />  
  );  
}
```



Cardápio

CategoriaGrid.js

```
<View style={styles.itemGrid}>
```

```
const styles = StyleSheet.create({  
  itemGrid: {  
    flex: 1,  
    margin: 16,  
    height: 150,  
    borderRadius: 8,  
    elevation: 4,  
    backgroundColor: "white",  
  },  
});
```

Cardápio

CategoriaGrid.js

```
<View style={styles.containerInterno}>
```

```
  containerInterno: {
```

```
    flex: 1,
```

```
    padding: 16,
```

```
    justifyContent: "center",
```

```
    alignItems: "center",
```

```
  },
```

Cardápio

CategoriaGrid.js

```
<Pressable style={styles.botao}>
```

```
  botao: {  
    flex: 1,  
  },
```

Cardápio

CategoriaGrid.js

```
<Text style={styles.titulo}>{titulo}</Text>
```

```
  titulo: {  
    fontSize: 18,  
  },
```

Cardápio

CategoriaGrid.js

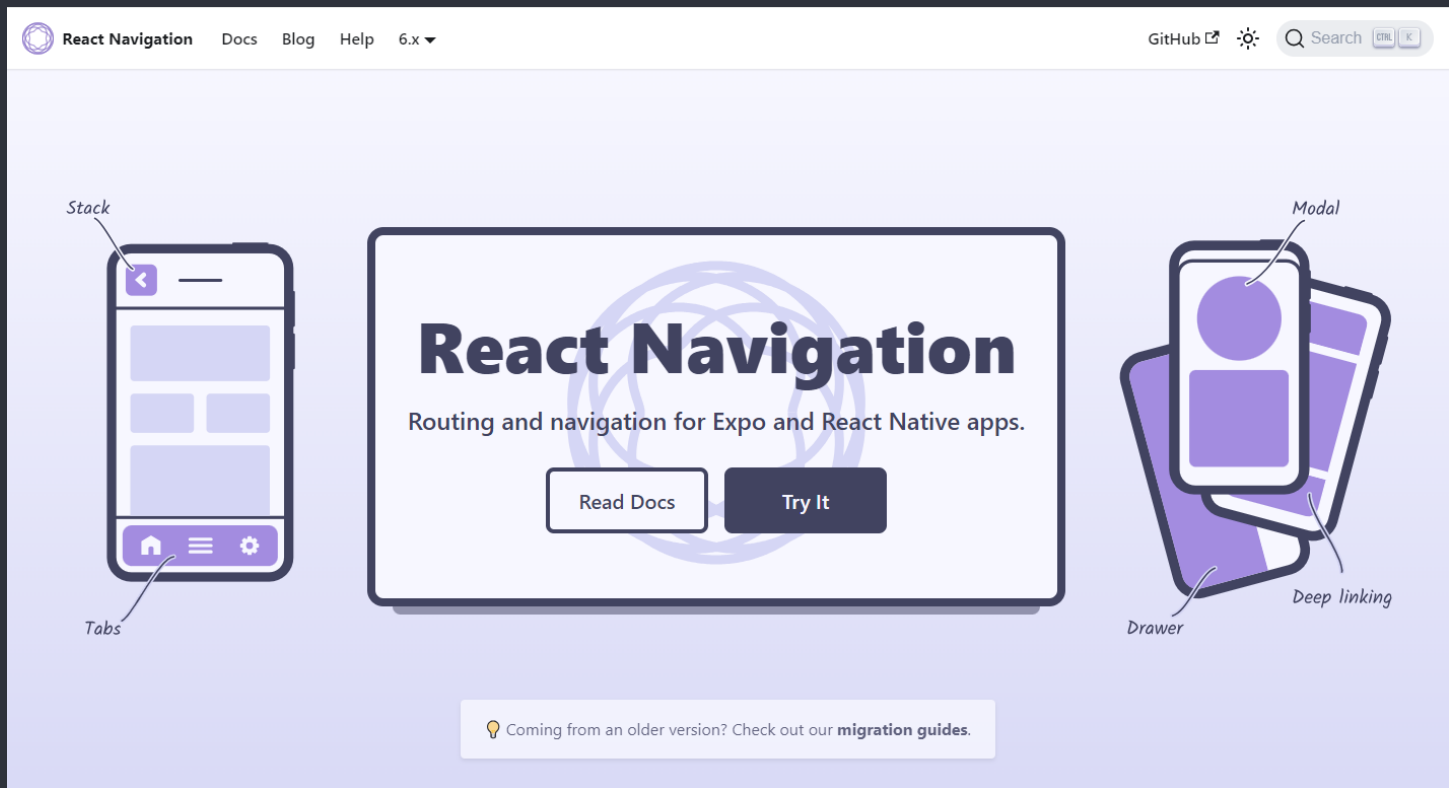
```
<Pressable android_ripple={{ color: "#CCC" }} style={styles.botao}>
```

```
    itemGrid: {  
      overflow: "hidden",  
    }
```

Cardápio

CategoriaGrid.js

```
<View style={[styles.itemGrid, { backgroundColor: cor }]}>
```



React Navigation

```
npm install @react-navigation/native
```

```
npm install react-native-screens react-native-safe-area-context
```

```
}
```


React Navigation

É uma biblioteca baseada em componentes

Nos fornece vários componentes que nos permitem configurar a navegação em aplicativos React Native

}

NavigationContainer

É um componente que deve envolver todos os componentes que irão possuir navegação no app

Geralmente todo o seu app

NavigationContainer

App.js

```
import { NavigationContainer } from "@react-navigation/native";
```

```
<NavigationContainer>  
  <CategoriaTela />  
</NavigationContainer>
```

React Navigation

Navigators

Stack

Native Stack

Drawer

Bottom Tabs

Material Bottom Tabs

Material Top Tabs

React Navigation

```
1  
2  
3  
4  
5  
6  
7  
8   npm install @react-navigation/native-stack  
9  
10  
11  
12  
13 }  
14
```

React Navigation

Vamos criar um “navigator” e registrar as telas que irão fazer parte dessa navegação

}

React Navigation

```
createNativeStackNavigator()
```


Retorna dois componentes:

- Navigator
- Screen

```
}
```

React Navigation

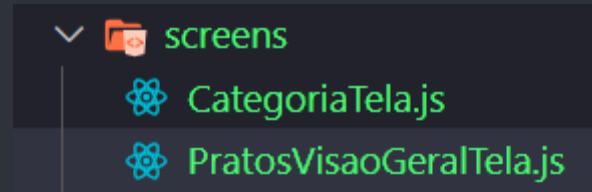
```
1
2
3 import { createNativeStackNavigator } from "@react-navigation/native-stack";
4
5 const Stack = createNativeStackNavigator();
6
7 export default function App() {
8   return (
9     <NavigationContainer>
10      <Stack.Navigator>
11        <Stack.Screen />
12      </Stack.Navigator>
13    </NavigationContainer>
14  );
15 }
```



React Navigation

```
<Stack.Screen name="Categorias" component={CategoriaTela} />
```

React Navigation



React Navigation

```
1  function PratosVisaoGeralTela() {  
2  
3      return (<View style={styles.container}>  
4          <Text>Tela de visão geral dos pratos</Text>  
5          </View>)  
6  }  
7  
8  export default PratosVisaoGeralTela;  
9  
10 const styles = StyleSheet.create({  
11     container: {  
12         flex: 1,  
13         padding: 16  
14     }  
});
```

React Navigation

App.js

```
<Stack.Screen name="PratosVisaoGeral" component={PratosVisaoGeralTela} />
```

React Navigation

CategoriaGid.js



```
1 function CategoriaGrid({ titulo, cor, onPress }) {  
2   return (  
3     <View style={[styles.itemGrid, { backgroundColor: cor }]}>  
4       <Pressable  
5         android_ripple={{ color: "#CCC" }}  
6         style={styles.botao}  
7         onPress={onPress} ←  
8       >  
9         <View style={styles.containerInterno}>  
10           <Text style={styles.titulo}>{titulo}</Text>  
11         </View>  
12       </Pressable>  
13     </View>  
14   );  
}
```

React Navigation

CategoriaGid.js



```
1 function CategoriaGrid({ titulo, cor, onPress }) {  
2   return (  
3     <View style={[styles.itemGrid, { backgroundColor: cor }]}>  
4       <Pressable  
5         android_ripple={{ color: "#CCC" }}  
6         style={styles.botao}  
7         onPress={onPress} ←  
8       >  
9         <View style={styles.containerInterno}>  
10           <Text style={styles.titulo}>{titulo}</Text>  
11         </View>  
12       </Pressable>  
13     </View>  
14   );  
}
```

React Navigation

CategoriaTela.js

```
function irParaTelaVisaoGeralPratos() {}
```

```
return (
```

```
  <CategoriaGrid
```

```
    titulo={itemData.item.titulo}
```

```
    cor={itemData.item.cor}
```

```
    onPress={irParaTelaVisaoGeralPratos}
```

```
  />
```

```
);
```



React Navigation

Nos componentes que utilizamos como telas, o React Navigator fornece uma propriedade especial para navegação:

`navigation`

```
<Stack.Screen name="Categorias" component={CategoriaTela} />  
<Stack.Screen name="PratosVisaoGeral" component={PratosVisaoGeralTela} />
```


1 React Navigation

2
3 CategoriaTela.js

4
5
6
7
8 `function CategoriaTela({ navigation }) {`



React Navigation

CategoriaTela.js

```
function irParaTelaVisaoGeralPratos() {  
    navigation.navigate("PratosVisaoGeral");  
}
```

Tela padrão

Ao configurar um Navigator (como `<Stack.Navigator>`) e cadastrar suas telas (via `<Stack.Screen>`), você pode decidir qual tela será mostrada como padrão quando o aplicativo for iniciado.

A tela superior (ou seja, o primeiro filho dentro de `<Stack.Navigator>`) é usada como tela inicial.

```
1 Stack.Screen
```

```
2
```

```
3
```

```
4
```

```
5
```

```
options
```

```
6
```

```
title
```

```
7
```

```
8
```

```
9 String que pode ser usada como substituto para  
10 headerTitle
```

```
11
```


```
12
```

```
13
```

```
14
```

Stack.Screen

```
1
2
3
4
5      <Stack.Screen
6          name="PratosVisaoGeral"
7          component={PratosVisaoGeralTela}
8          options={{
9              title: "Pratos",
10             }}
11      />
```




Passagem de parâmetros entre as telas

CategoriaTela.js

```
navigation.navigate("PratosVisaoGeral", {  
  categoriaId: itemData.item.id,  
});
```

Passagem de parâmetros entre as telas

PratosVisaoGeral.js



```
function PratosVisaoGeralTela({ navigation, route })  
{  
  const categoriaId = route.params.categoriaId;  
  ...  
}
```

```
new Prato(  
  1  "m2",  
    ["c3"],  
  2  "Hambúrguer",  
    "barato",  
  3  "https://cdn.pixabay.com/photo/2014/10/23/18/05/burger-500054_1280.jpg",  
    [  
      4  "300g de carne de boi",  
        "1 tomate",  
        5  "1 Pepino",  
        "1 Cebola",  
        6  "Ketchup",  
        "1 pães de hambúrguer",  
      7  ],  
      8  [  
        9  "Forme 2 bifos de hambúrguer",  
        10  "Frite os hambúrgueres por 4 minutos de cada lado",  
        11  "Frite rapidamente os pães por 1 minuto de cada lado",  
        12  "Passe ketchup no pães",  
        "Sirva o hambúrguer com tomate, pepino e cebola",  
      13  ],  
      false,  
      14  true  
    )  
)
```



```
new Prato(  
  1  "m3",  
  2  ["c3"],  
  3  "X-Buguer",  
  4  "barato",  
  5  "https://burgerx.com.br/assets/img/galeria/burgers/x-salada.jpg",  
  6  [  
  7    "300g de carne de boi",  
  8    "1 tomate",  
  9    "2 fatias de mussarela",  
 10    "1 folha de alface",  
 11    "Ketchup",  
 12    "1 pães de hambúrguer",  
 13  ],  
  14  [  
    "Frite os hambúrgueres com um pouquinho de óleo , quando estiver  
    frito cubra com a mussarela.",  
    "Retire do fogo e coloque no pão",  
    "Passe no pão a maionese e coloque o alface e tomate",  
  ],  
  false,  
  false  
)
```

Filtrando os pratos

PratosVisaoGeral.js

```
const pratosVisiveis = PRATOS.filter((itemPrato) => {  
  return itemPrato.categoriaIds.indexOf(categoriaId) >= 0;  
});
```

Filtrando os pratos

PratosVisaoGeral.js

```
return (  
  <View style={styles.container}>  
    <FlatList  
      data={pratosVisiveis}  
      keyExtractor={(item) => item.id}  
      renderItem={renderItemPrato}  
    />  
  </View>  
);
```

Filtrando os pratos

PratosVisaoGeral.js

```
function renderItemPrato(itemData) {  
  return (  
    <View>  
      <Text>{itemData.item.titulo}</Text>  
    </View>  
  );  
}
```

1 Filtrando os pratos

2

3

4

5

6

7

8

9

10

11

12

13

14



Filtrando os pratos

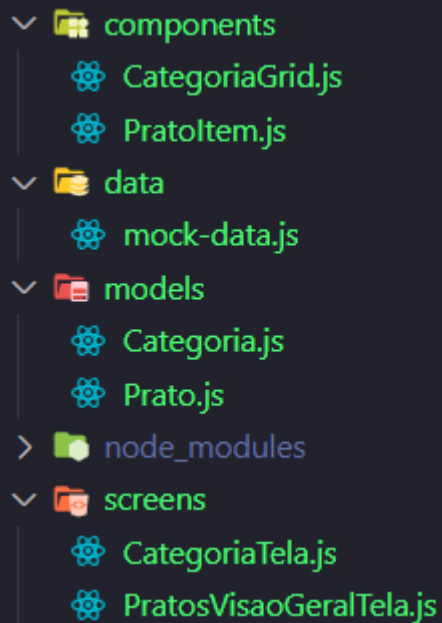
```
function PratoItem({ titulo }) {  
  return (  
    <View>  
      <Text>{titulo}</Text>  
    </View>  
  );  
}  
  
export default PratoItem;
```

Filtrando os pratos

PratosItem.js

```
const styles = StyleSheet.create({  
  titulo: {  
    textAlign: "center",  
    fontSize: 18,  
  },  
});
```

Cardápio



```
▼ components
  ├── CategoriaGrid.js
  └── PratoItem.js
▼ data
  └── mock-data.js
▼ models
  ├── Categoria.js
  └── Prato.js
> node_modules
▼ screens
  ├── CategoriaTela.js
  └── PratosVisaoGeralTela.js
```


1 Estilizando tela de pratos

```
2  
3  
4  
5 function PratoItem({ titulo }) {  
6   return (  
7     <View>  
8       <Text style={styles.titulo}>{titulo}</Text>  
9     </View>  
10  );  
11 }  
12  
13  
14
```

Filtrando os pratos

PratosItem.js

```
function PratoItem({ titulo, urlImagem }) {  
  return (  
    <View>  
      <Pressable>  
        <View>  
          <Image source={{ uri: urlImagem }} style={styles.imagem} />  
          <Text style={styles.titulo}>{titulo}</Text>  
        </View>  
      </Pressable>  
    </View>  
  );  
}
```

1 Filtrando os pratos


2 PratosItem.js

```
3
4 const styles = StyleSheet.create({
5   imagem: {
6     width: "100%",
7     height: 200,
8   },
9   titulo: {
10    textAlign: "center",
11    fontSize: 18,
12    margin: 8,
13  },
14 });
```

1 Filtrando os pratos

2 PratosVisaoGeralTela.js

```
3  
4  
5     function renderItemPrato(itemData) {  
6         return (  
7             <PratoItem  
8                 titulo={itemData.item.titulo}  
9                 urlImagem={itemData.item.urlImagem}  
10            />  
11        );  
12    }  
13  
14
```



1 Filtrando os pratos

2 PratosItem.js

3
4 <View>

5 <Pressable>

6 <View>

7 <Image source={{ uri: urlImagem }} style={styles.imagem} />

8 <Text style={styles.titulo}>{titulo}</Text>

9 </View>

10 <View>

11 <Text>{acessibilidadePreco.toUpperCase()}</Text>

12 </View>

13 </Pressable>


14 </View>



Filtrando os pratos

PratosVisaoGeralTela.js

```
function renderItemPrato(itemData) {  
  return (  
    <PratoItem  
      titulo={itemData.item.titulo}  
      urlImagem={itemData.item.urlImagem}  
      acessibilidadePreco={itemData.item.acessibilidadePreco}  
    />  
  );  
};
```



Filtrando os pratos

PratosItem.js

```
pratoItem: {  
  margin: 16,  
  borderRadius: 8,  
  backgroundColor: "white",  
  overflow: "hidden",  
  elevation: 4,  
},  
detalhes: {  
  padding: 8,  
  alignItems: "center",  
},
```

Filtrando os pratos

PratosItem.js



```
<View style={styles.pratoItem}>
  <Pressable>
    <View>
      <Image source={{ uri: urlImagem }} style={styles.imagem} />
      <Text style={styles.titulo}>{titulo}</Text>
    </View>
    <View style={styles.detalhes}>
      <Text>{acessibilidadePreco.toUpperCase()}</Text>
    </View>
  </Pressable>
</View>
```



Filtrando os pratos

PratosItem.js

```
<Pressable android_ripple={{ color: "#00000088" }}>
```

Alterando o título da página

PratosVisaoGeralTela.js

```
import { PRATOS, CATEGORIAS } from "../data/mock-data";
```

```
const categoriaTitulo = CATEGORIAS.find(  
  (categoria) => categoria.id == categoriaId  
)  
.titulo;  
console.log(categoriaTitulo);
```

1 Alterando o título da página

2

3 PratosVisaoGeralTela.js

4

5

6

7

8

9

10

11

12

13

14

```
navigation.setOptions({  
  title: categoriaTitulo,  
});
```

1 useEffect hook

2
3
4
5 O useEffect é um Hook que serve para lidar com os
6 efeitos colaterais no componentes.
7

8
9 Como atualizar a tela com dados retornados de uma
10 API ou atualizar o título da tela.
11
12
13
14

useEffect hook

```
1  useEffect hook
2
3  useEffect(() => {
4      const categoriaTitulo = CATEGORIAS.find(
5          (categoria) => categoria.id == categoriaId
6      ).titulo;
7      console.log(categoriaTitulo);
8
9
10     navigation.setOptions({
11         title: categoriaTitulo,
12     });
13 });
14
```

Tela de detalhes do prato



1 Tela de detalhes do prato

```
2  
3   import { Text } from "react-native";  
4  
5  
6   function PratoDetalheTela() {  
7     return <Text>Detalhe do prato</Text>;  
8  
9   }  
10  
11  
12   export default PratoDetalheTela;  
13  
14
```

1 Tela de detalhes do prato

2
3 App.js

4

5

6

7

8

```
<Stack.Screen name="PratoDetalhe" component={PratoDetalheTela} />
```

9

10

11

12

13

14

1 navigation

2
3
4
5 Quando estamos em um componente registrado como
6 screen <Stack.Screen> temos disponível a
7 propriedade *navigation* para navegar entre as telas.
8

9 Assim fizemos na tela de categorias.
10
11
12
13
14

```
1 navigation
2
3
4
5 function irParaTelaVisaoGeralPratos() {
6     navigation.navigate("PratosVisaoGeral", {
7         categoriaId: itemData.item.id,
8     });
9 }
10
11
12
13
14
```

1 navigation

2

3

4

5

6 Mas se estivermos dentro de um componente que não é
7 registrado como tela e quisermos navegar para uma
8 tela, como podemos fazer?

9

10

11

12

13

14

1 navigation

2

3

4

5

6

7

8

9

10

11

12

13

14

PratoItem.js



PratoDetalheTela.js

1 navigation

2

3 PratoItem.js

4

5

6

7 const navigation = useNavigation();

8

9 navigation.navigate('PratoDetalhe');

10

11

12

13

14

1 navigation

2

3 PratoItem.js

4



5

function PratoItem({ id, titulo, urlImagem, acessibilidadePreco }

6

{

7

const navigation = useNavigation();

8

9

navigation.navigate('PratoDetalhe', {

10

pratoId: id

11

});

12

13

14

1 navigation

2

3 PratosVisaoGeralTela.js

4

5

6

7

`<PratoItem`

8

`id={itemData.item.id}`

9

`titulo={itemData.item.titulo}`

10

`urlImagem={itemData.item.urlImagem}`

11

`acessibilidadePreco={itemData.item.acessibilidadePreco}`

12

`/>`

13

14

```
1 navigation
2
3 PratoItem.js
4
5 function irParaTelaDeDetalhe() {
6     navigation.navigate("PratoDetalhe", {
7         pratoId: id,
8     });
9 }
10
11 <Pressable
12     android_ripple={{ color: "#00000088" }}
13     onPress={irParaTelaDeDetalhe}
14 >
```


1 navigation

2

3 PratoDetalheTela.js

4

5

6

```
7 function PratoDetalheTela({ route }) {  
8     const pratoId = route.params.pratoId;  
9     return <Text>Detalhe do prato - {pratoId}</Text>;  
10 }  
11
```

12

13

14

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5 return (

6

7 <View>

8

9 <Image />

10

11 <Text></Text>

12

13 <View></View>

14

15 <Text>Ingredientes</Text>

16

17 <Text>Passo a passo</Text>

18

19

20 </View>

21

22);

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5

```
6 const pratoSelecionado = PRATOS.find((prato) => prato.id == pratoId);
```

7

```
8 console.log(pratoSelecionado);
```

9

10

11

12

13

14

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

`<Image`

5

`source={{ uri: pratoSelecionado.urlImagem }}`

6

`style={styles.imagem}`

7

`/>`

8

9

`const styles = StyleSheet.create({`

10

 `imagem: {`

11

 `width: "100%",`

12

 `height: 350,`

13

 `},`

14

`});`

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5 <Text>{pratoSelecionado.titulo}</Text>

6 <View style={styles.detalhes}>

7 <Text>{pratoSelecionado.acessibilidadePreco.toUpperCase()}</Text>

8 </View>

9

10

11 detalhes: {

12 padding: 8,

13 alignItems: "center",

14 },

Tela de detalhe

PratoDetalheTela.js

```
<Text>Ingredientes</Text>
{pratoSelecionado.ingredientes.map((ingrediente) => (
  <Text key={ingrediente}>{ingrediente}</Text>
))}

<Text>Passo a passo</Text>
{pratoSelecionado.passos.map((passo) => (
  <Text key={passo}>{passo}</Text>
))}
```

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5

6

7

titulo: {

8

fontSize: 24,

9

10

margin: 8,

11

textAlign: "center",

12

},

13

14

1 Tela de detalhe

2

3 PratoDetalheTela.js



4

```
<Text style={styles.subtitulo}>Ingredientes</Text>
```

5

6

7

```
subtitulo: {
```

8

```
  fontSize: 18,
```

9

```
  margin: 6,
```

10

```
  fontWeight: "bold",
```

11

```
  textAlign: "center",
```

12

```
  color: "#555",
```

13

```
},
```

14

1 Tela de detalhe

2

3

4

5

6

7

8

9

10

11

12

13

14



1 Tela de detalhe

```
2  
3  
4  
5  
6 function Lista() {  
7     return pratoSelecionado.ingredientes.map((ingrediente) => (  
8         <Text key={ingrediente}>{ingrediente}</Text>  
9     ));  
10 }  
11  
12  
13  
14
```

1 Tela de detalhe

```
2  
3  
4 function Lista({ dados }) {  
5   return dados.map((dado) => (  
6     <View key={dado}>  
7       <Text>{dado}</Text>  
8     </View>  
9   ));  
10 }  
11  
12  
13  
14
```

1 Tela de detalhe

```
2   const styles = StyleSheet.create({
3     itemLista: {
4       borderRadius: 6,
5       paddingHorizontal: 8,
6       paddingVertical: 4,
7       marginVertical: 4,
8       marginHorizontal: 12,
9       backgroundColor: "#555",
10    },
11    itemTexto: {
12      color: "#FFFFFF",
13      textAlign: "center",
14    },
15  });
```

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5

6

7

8

9 `<Lista dados={pratoSelecionado.ingredientes} />`

10

11

12

13

14

1 Tela de detalhe

2

3 PratoDetalheTela.js

4

5

6

7 return (
8 <ScrollView>
9

10

11 ...

12

13

14