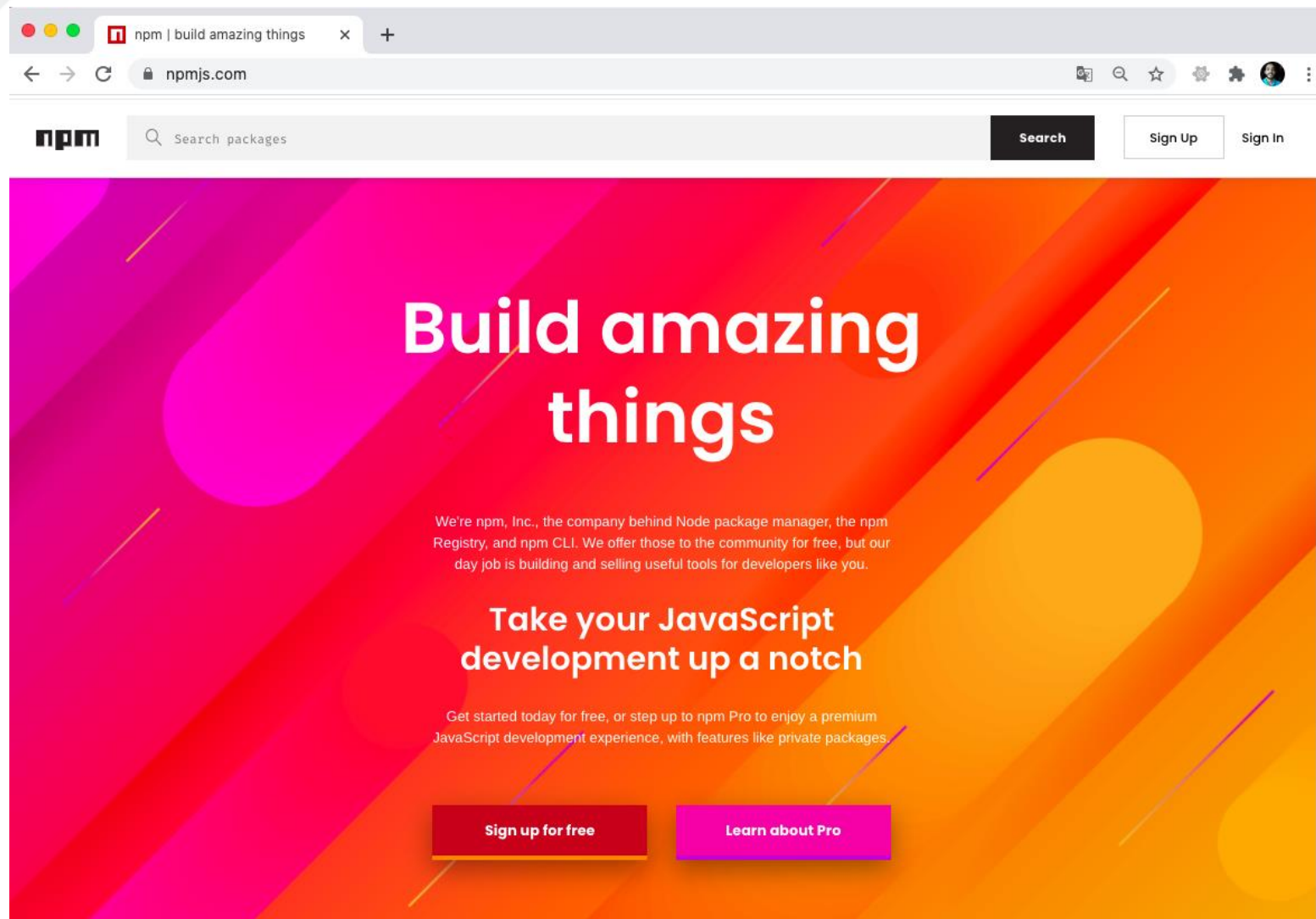


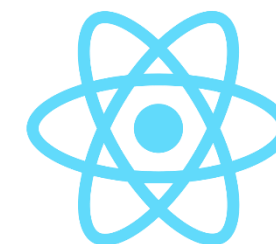


(3.1.1) Persistência de Dados e Navegação

Instalando bibliotecas em um projeto



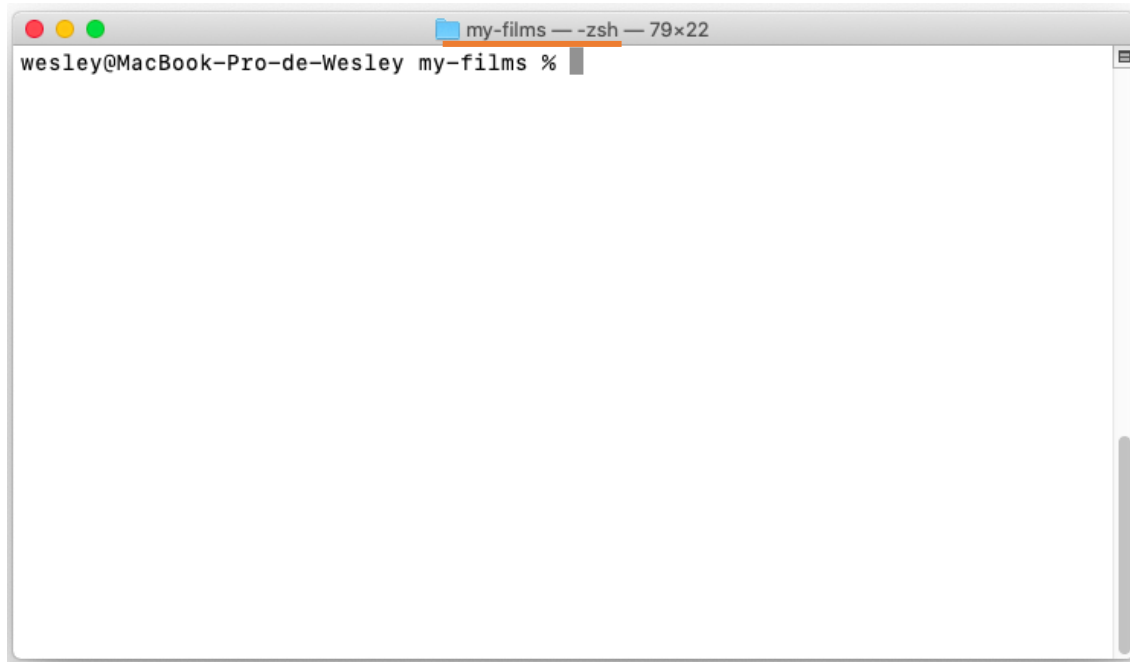
<https://www.npmjs.com/>



Instalando bibliotecas em um projeto

Para instalar bibliotecas:

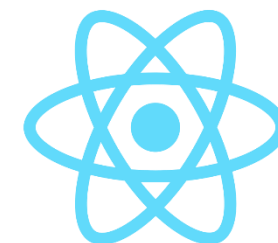
1 - Acesse o “path” do projeto através do terminal:

A screenshot of a macOS terminal window. The title bar at the top shows a folder icon, the text 'my-films', and a status bar with '- zsh' and '79x22'. The terminal content shows the prompt 'wesley@MacBook-Pro-de-Wesley my-films %' with a cursor at the end. The window has standard macOS window controls (red, yellow, green buttons) in the top-left corner.

```
my-films -- zsh -- 79x22
wesley@MacBook-Pro-de-Wesley my-films %
```

2 - Execute o comando de instalação da biblioteca:

- `npm install react-native-smart-page`





Obrigado!



(3.2.1) Persistência de Dados I

Persistência de dados

Persistência de dados garante que um dado foi salvo e que poderá ser recuperado e assim utilizado em alguma situação. O React Native oferece um banco de dados padrão para ser usado, o **Async Storage**.

@react-native-community/async-storage
1.11.0 • Public • Published 3 months ago

Readme Explore BETA 1 Dependency 235 Dependents 37 Versions

React Native Async Storage

An asynchronous, unencrypted, persistent, key-value storage system for React Native.

Supported platforms

- iOS
- Android
- Web
- MacOS
- Windows

Getting Started

Install

```
> npm i @react-native-community/async-storage
```

Weekly Downloads

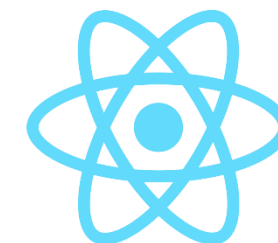
247.612

Version	License
1.11.0	MIT

Unpacked Size	Total Files
291 kB	93

Homepage
github.com/react-native-community/re...

<https://www.npmjs.com/package/@react-native-community/async-storage>

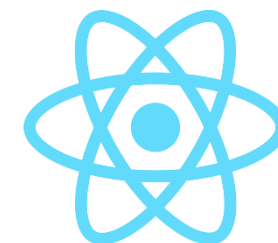


Persistência de dados – Async Storage

Async Storage é uma API nativa do React Native, utilizada para armazenar dados persistentes no dispositivo. É uma forma de salvar dados no formato chave e valor. Os dados salvos com a API Async Storage são assíncronos, com isso retornam um Promise, e em caso de erro retorna um Error.

As ações básicas mais utilizadas com o Async Storage são:

- Gravar um item
- Recuperar um item e seu valor
- Remover um item

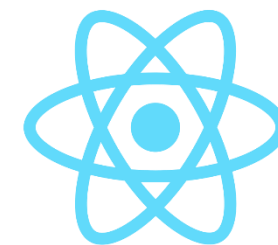


Persistência de dados – Async Storage

Gravando item

Para gravar dados no Async Storage, você pode utilizar o método `setItem()`. Este método é usado para adicionar novos itens de dados (quando não existem dados para uma determinada chave) e para modificar o item, caso já exista determinada chave.

```
const storeData = async () => {  
  try {  
    await AsyncStorage.setItem('@storage_Key', value)  
  } catch (e) {  
    // saving error  
  }  
}
```

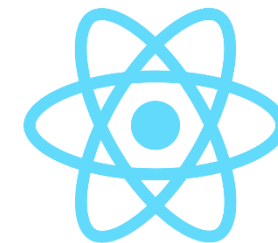


Persistência de dados – Async Storage

Recuperando item

Para recuperar dados no Async Storage, você pode utilizar o método `getItem()`. Este método retorna uma promessa que ao ser resolvida devolve um valor encontrado através da sua chave pesquisada. Caso não encontre a chave, retorna `null`.

```
getData = async () => {  
  try {  
    const value = await AsyncStorage.getItem('@storage_Key')  
    if (value !== null) {  
      // value previously stored  
    }  
  } catch (e) {  
    // error reading value  
  }  
}
```

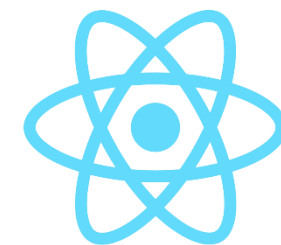


Persistência de dados – Async Storage

Removendo item

Para deletar um item do storage, utilizamos o método:

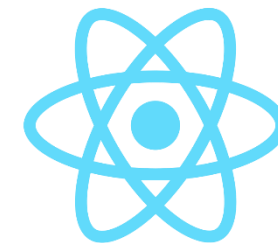
```
deleteData = async () => {  
  try {  
    await AsyncStorage.removeItem('@storage_Key ');  
  } catch (error) {  
    console.log(error.message);  
  }  
}
```



Persistência de dados – Async Storage

Componente utilizando métodos do Async Storage

```
import React from 'react';
import { View } from 'react-native';
import AsyncStorage from '@react-native-community/async-storage';
const App = () => {
  const storeData = async () => {
    try {
      const jsonValue = JSON.stringify(value)
      await AsyncStorage.setItem('@storage_Key', jsonValue)
    } catch (e) {
      // saving error
    }
  }
  return (
    <View>
      <Text>Hello, World!</Text>
    </View>
  )
}
export default App;
```





Obrigado!



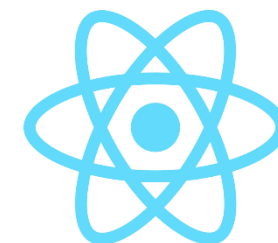
(3.2.2) Persistência de Dados II

Persistência de dados – Async Storage

Gravando item

Async Storage só permite armazenar dados do tipo string, portanto, para armazenar dados de objetos, você precisa serializá-los primeiro. Para serializar dados, você pode usar `JSON.stringify()`.

```
const storeData = async () => {  
  try {  
    const jsonValue = JSON.stringify(value)  
    await AsyncStorage.setItem('@storage_Key', jsonValue)  
  } catch (e) {  
    // saving error  
  }  
}
```

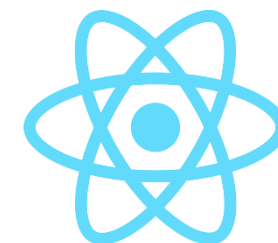


Persistência de dados – Async Storage

Recuperando item

Para recuperar os dados que foram serializados, devemos utilizar o método `JSON.parse()` e assim voltarmos ele ao seu tipo original.

```
getData = async () => {  
  try {  
    const jsonValue = await AsyncStorage.getItem('@storage_Key')  
    return jsonValue != null ? JSON.parse(jsonValue) : null;  
  } catch (e) {  
    // error reading value  
  }  
}
```





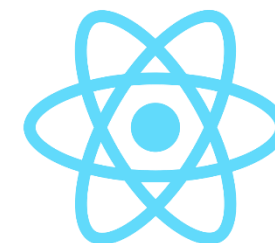
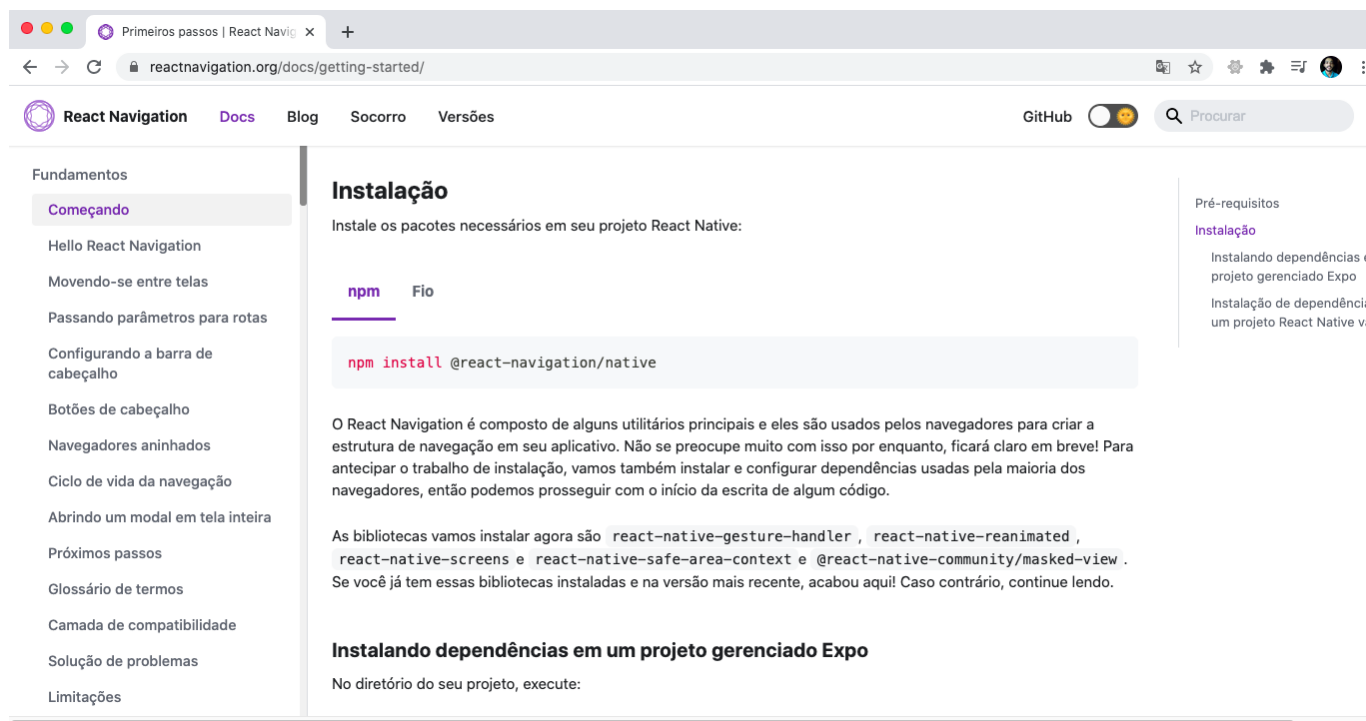
Obrigado!



(3.3.1) Navegação entre telas

Navegação entre Telas

Um dos recursos mais básicos em qualquer aplicação é a possibilidade de navegar e interagir entre diversas telas de um app. Há várias maneiras de implementar essa funcionalidade, mas a biblioteca de maior destaque é a React Navigation.



Navegação entre Telas

Para configurar o React Navigation em um Projeto, deve ser executar o seguinte comando:

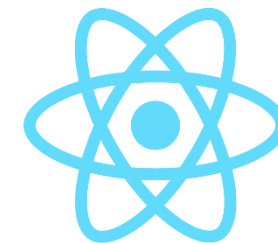
- `npm install @react-navigation/native`

O comando acima, depende de outras bibliotecas para funcionar corretamente. São elas: [react-native-gesture-handler](#), [react-native-reanimated](#), [react-native-screens](#), [react-native-safe-area-context](#) e [@react-native-community/masked-view](#).

Para instalar todos eles de uma vez, execute o seguinte comando:

```
npm install react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-view
```

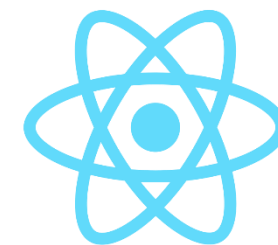
<https://reactnavigation.org/docs/getting-started/>



Navegação entre Telas

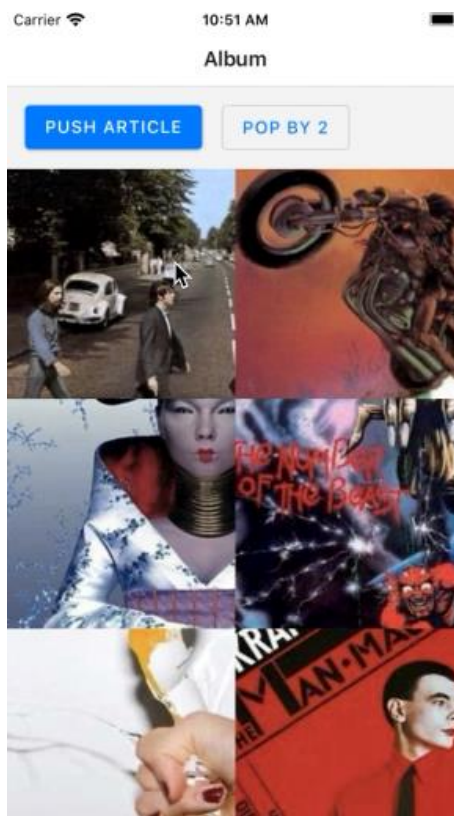
Existem diversos tipos de navegação entre telas em uma aplicação. Cada um desses tipos possui um objetivo e um tipo diferente de implementação, e são usados conforme a necessidade do projeto.

- **StackNavigator:** Utiliza uma navegação entre telas por meio de botões dentro das telas.
- **TabNavigator:** Utiliza uma navegação através de abas (tabs).
- **DrawerNavigator:** Utiliza uma navegação através um menu lateral.



Navegação entre Telas

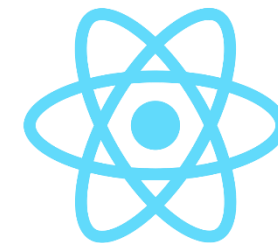
Stack Navigator



Por padrão, o navegador de pilha é configurado para ter a aparência e comportamento familiares do iOS e do Android: novas telas deslizam da direita no iOS e aparecem da parte inferior no Android. No iOS, o navegador de pilha também pode ser configurado para um estilo modal em que as telas deslizam da parte inferior.

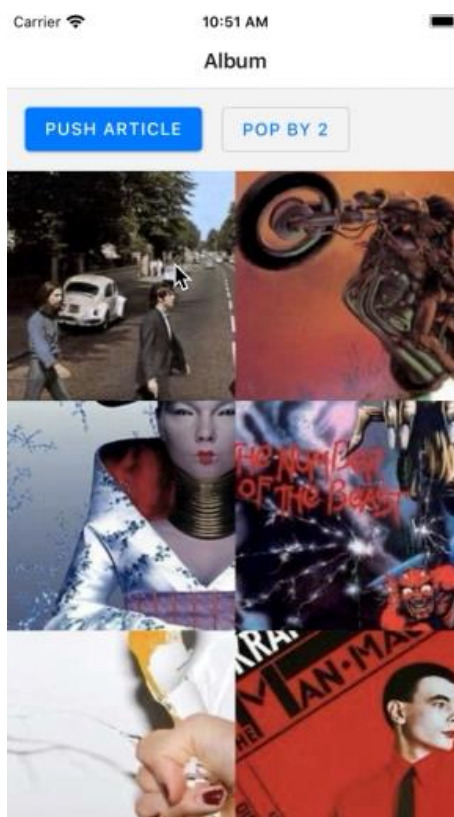
Para usar este navegador, certifique-se de que você possui [@react-navigation/native](#) e instale:

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



Navegação entre Telas

Stack Navigator

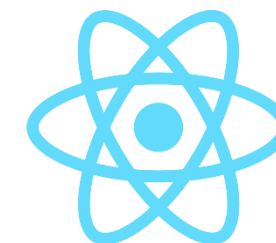


Para usar este navegador, importe-o @react-navigation/stack

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

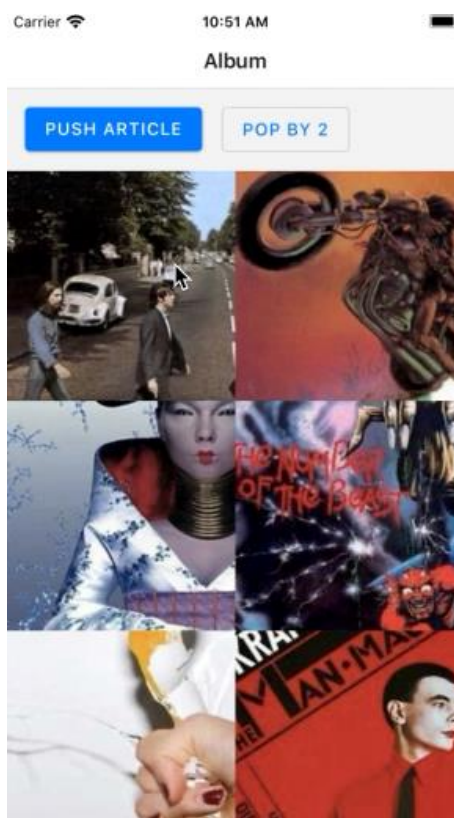
const Stack = createStackNavigator();
function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Notifications" component={Notifications} />
      <Stack.Screen name="Profile" component={Profile} />
      <Stack.Screen name="Settings" component={Settings} />
    </Stack.Navigator>
  );
}

export default MyStack;
```



Navegação entre Telas

Stack Navigator



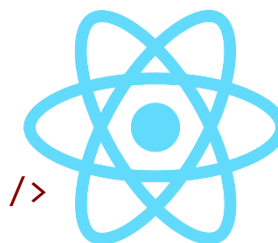
Para navegar entre telas, utilize o método `navigation.navigate('')` para abrir uma nova tela, e o método `navigation.goBack()` para retroceder a tela anterior.

```
import { useNavigation } from '@react-navigation/native';

function ProfileScreen() {

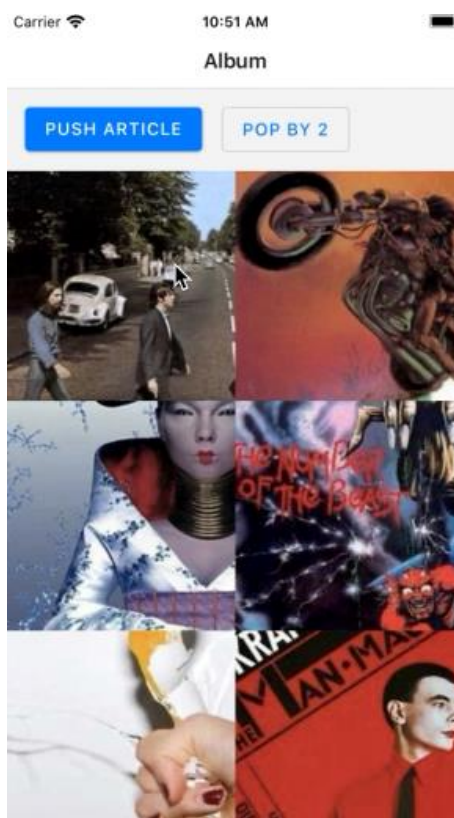
  const navigation = useNavigation();

  return (
    <Button
      title="Go to Notifications"
      onPress={() => navigation.navigate('Notifications')}
    />
    <Button title="Go back" onPress={() => navigation.goBack()} />
  );
}
```



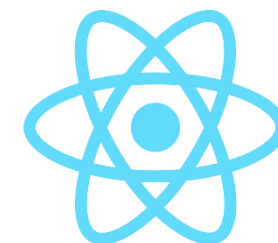
Navegação entre Telas

Stack Navigator - Estilização



Para estilizar a navegação, utilize a prop “options”, por exemplo:

```
<HomeStack.Navigator>
  <HomeStack.Screen
    name="A"
    component={A}
    options={{ tabBarLabel: 'Home!' }}
  />
</HomeStack.Navigator>
```





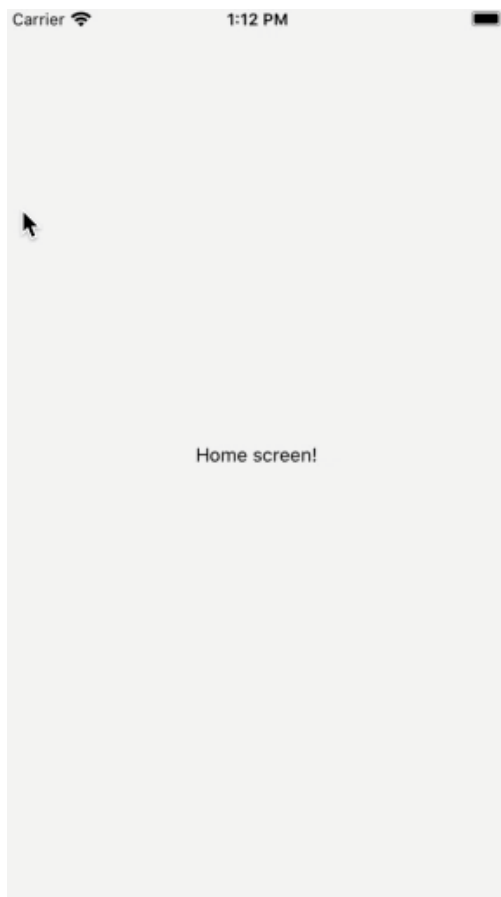
Obrigado!



(3.3.2) Drawer Navigator

Navegação entre Telas

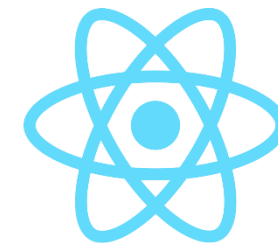
Drawer Navigator



O padrão comum nesse tipo de navegação é usar a gaveta do lado esquerdo (podendo ser o lado direito) para navegar entre as telas.

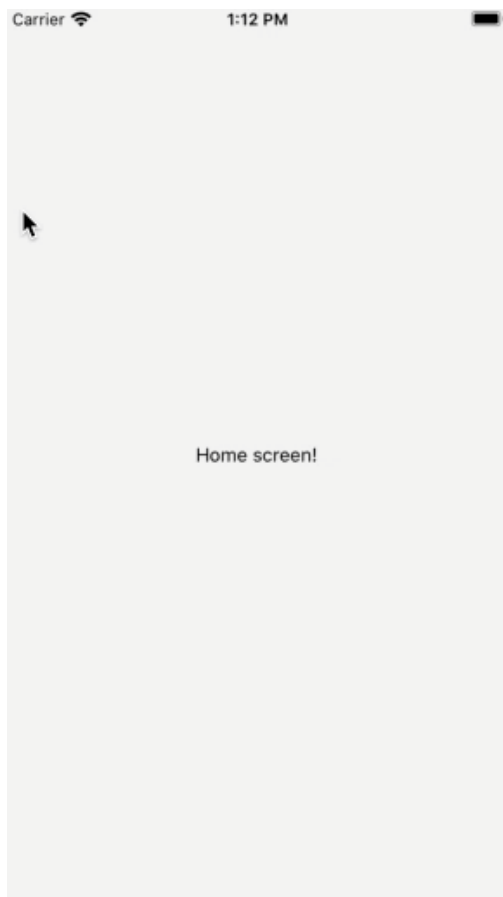
Para usar este navegador, certifique-se de que você possui [@react-navigation/native](#) e instale:

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



Navegação entre Telas

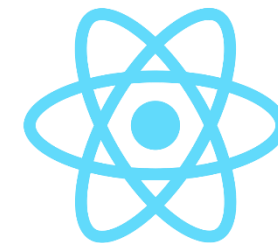
Stack Navigator



Para usar este navegador, importe-o `@react-navigation/stack`

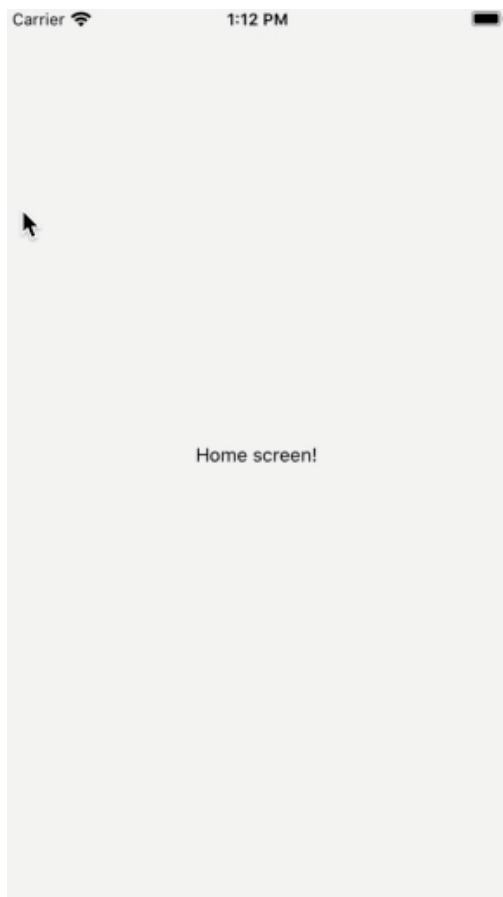
```
import { NavigationContainer } from '@react-navigation/native';
import { createDrawerNavigator } from '@react-navigation/drawer';

const Drawer = createDrawerNavigator();
export default function App() {
  return (
    <NavigationContainer>
      <Drawer.Navigator initialRouteName="Home">
        <Drawer.Screen name="Home" component={HomeScreen} />
        <Drawer.Screen name="Notifications" component={NotificationsScreen} />
      </Drawer.Navigator>
    </NavigationContainer>
  );
}
```



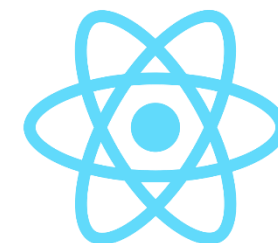
Navegação entre Telas

Stack Navigator - Estilização



Para estilizar a navegação, utilize a prop “options”, por exemplo:

```
<Drawer.Navigator
  drawerStyle={{
    backgroundColor: '#c6cbef',
    width: 240,
  }}
>
  {/* screens */}
</Drawer.Navigator>
```





Obrigado!



(3.3.3) Tab Navigator

Navegação entre Telas

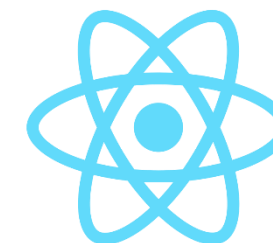
Tab Navigator



Possivelmente, o estilo mais comum de navegação em aplicativos móveis é a navegação baseada em guias.

Para usar este navegador, certifique-se de que você possui [@react-navigation/native](#) e instale:

```
npm install @react-navigation/bottom-tabs
```



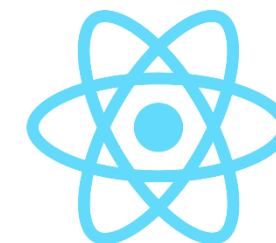
Navegação entre Telas

Tab Navigator



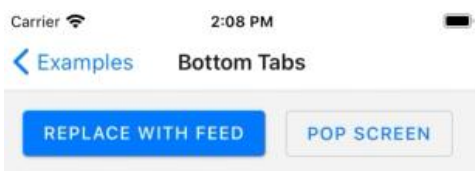
Para usar este navegador, importe-o `@react-navigation/bottom-tabs`

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
const Tab = createBottomTabNavigator();
function MyTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Home" component={HomeScreen} />
      <Tab.Screen name="Settings" component={SettingsScreen} />
    </Tab.Navigator>
  );
}
```



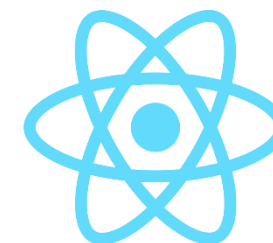
Navegação entre Telas

Tab Navigator - Estilização



Para estilizar a navegação, utilize a prop “options”, por exemplo:

```
<Tab.Screen
  name="Profile"
  component={Profile}
  options={{
    tabBarLabel: 'Profile',
    tabBarIcon: ({ color, size }) => (
      <MaterialCommunityIcons name="account" color={color} size={size} />
    ),
  }}
/>
```





Obrigado!



(3.3.4) Combinando Navegações

Navegação entre Telas

Combinando Navegações

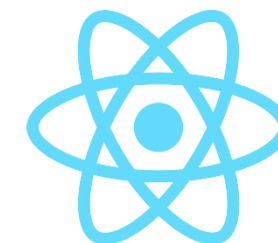


```
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
```

```
const AppTabs = createBottomTabNavigator();
```

```
const Tabs = () => (
  <AppTabs.Navigator>
    <AppTabs.Screen
      name="Home"
      component={Home}
    />
    <AppTabs.Screen
      name="Listagem"
      component={Listagem}
    />
  </AppTabs.Navigator>
);
```

```
export default () => {
  return (
    <AppStack.Navigator
      initialRouteName="Preload"
      screenOptions={{ headerShown: false }}>
      <AppStack.Screen
        name="Main" component={Main} />
      <AppStack.Screen
        name="Tabs" component={Tabs} />
    </AppStack.Navigator>
  );
};
```





Obrigado!



(3.3.5) Passando Propriedades entre telas

Passando Propriedades entre telas

Com o React Navigation, podemos compartilhar propriedades entre telas.

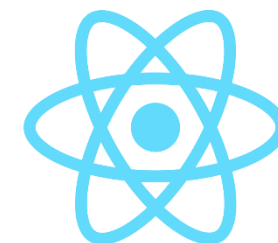
- O segundo parâmetro do `navigate` recebe o objeto a ser passado para a tela navegada.

```
navigation.navigate('Detail', { name: 'Wesley' })
```

- Utilize o objeto `useRoute()` do React Navigation para receber o parâmetro que foi passado.

```
import { useRoute } from '@react-navigation/native';
```

```
const App = () => {  
  
  const route = useRoute();  
  const name = route.params.name;  
  
  return <Text>{name}</Text>  
}  
  
export default App;
```



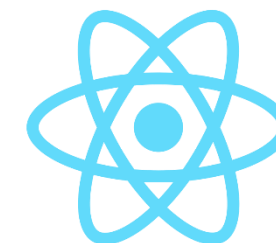
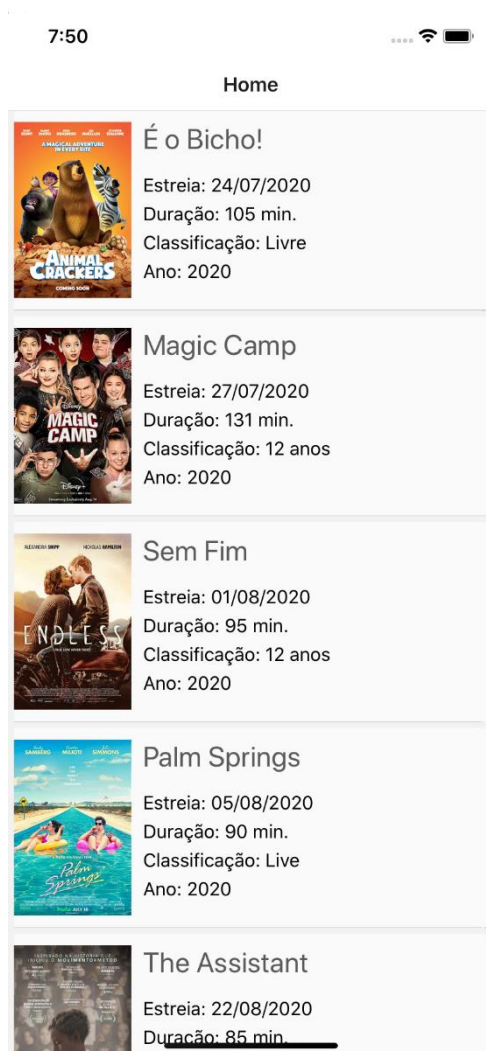


Obrigado!



(3.4.1) Projeto Prático

Projeto Prático - Filmes em Cartaz





Obrigado!