Workshop de introdução ao React Native

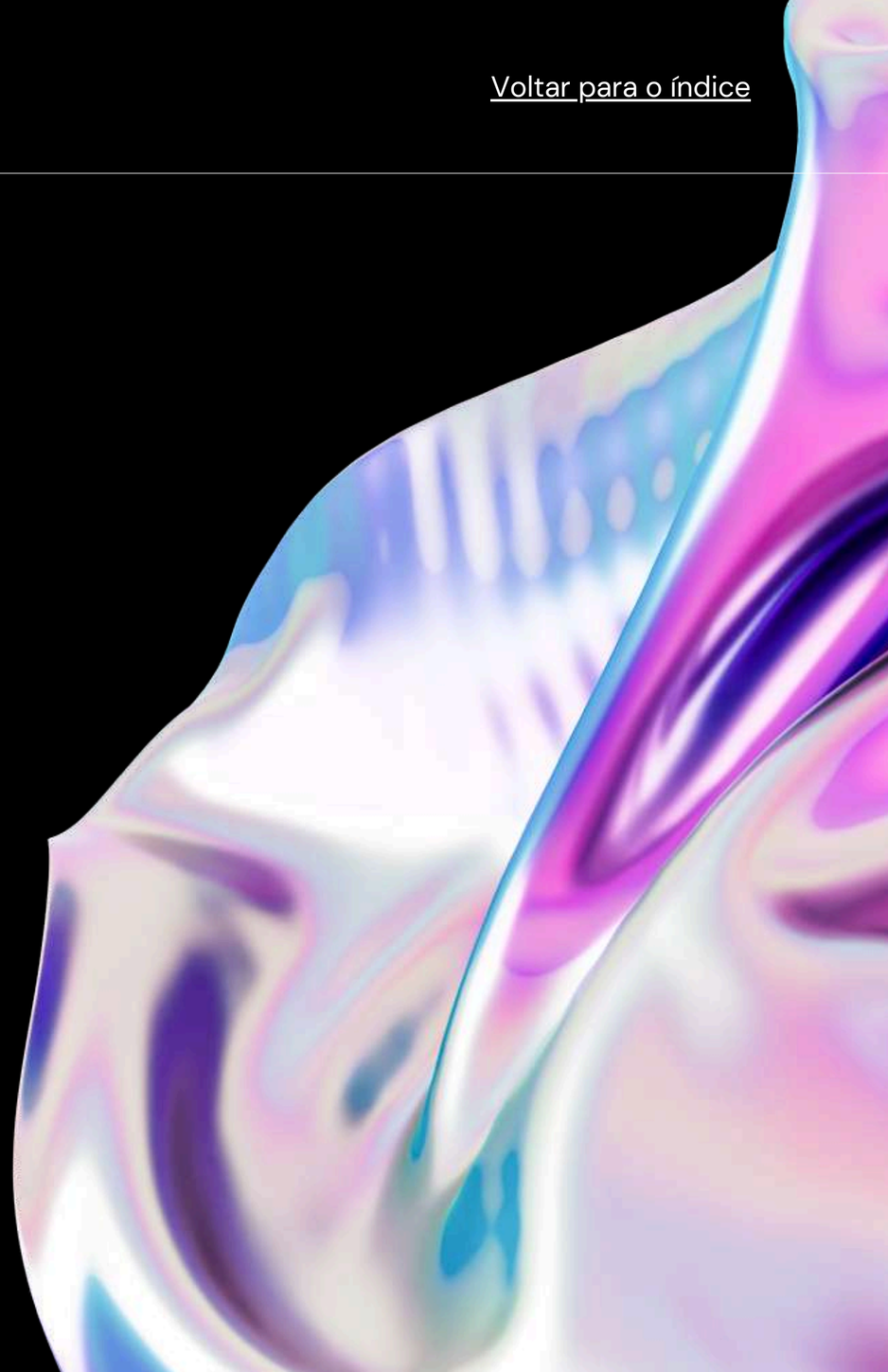
Criando um Aplicativo de Diário de Viagem
com React Native e Expo Go

Índice

Voltar para o índice

O que é React Native?

React Native é uma tecnologia de desenvolvimento de aplicativos móveis criada pelo Facebook. Ela permite que você construa aplicativos móveis para iOS e Android usando apenas JavaScript e React, que é uma biblioteca para construção de interfaces de usuário.



Voltar para o índice

Principais Características

Código Reutilizável

Você pode escrever um único código que funciona tanto em iOS quanto em Android, o que economiza tempo e esforço.

Desenvolvimento Rápido

Com React Native, você pode usar as mesmas ferramentas e conceitos que você já conhece se você já trabalha com React para web.

Componentes Nativos

React Native usa componentes nativos para criar a interface do usuário, o que significa que seu aplicativo terá a aparência e o desempenho de um aplicativo nativo.

Comunidade Ativa

Há uma grande comunidade de desenvolvedores que contribuem com bibliotecas e ferramentas, facilitando a resolução de problemas e a adição de funcionalidades ao seu aplicativo.

Voltar para o índice

O que é Expo Go e por que usá-lo?

Expo Go é uma ferramenta e um conjunto de serviços que simplificam o desenvolvimento com React Native. Ele é especialmente útil para iniciantes e para prototipagem rápida.

Voltar para o índice

Configuração do ambiente

Linux

Atualize os pacotes:

```
sudo apt update
```

Instale o Node.js e npm usando o gerenciador de pacotes oficial:

```
sudo apt install nodejs npm
```

Verifique a instalação

```
node -v
npm -v
```

Windows

Baixe o instalador do Node.js do site oficial.

Execute o instalador e siga as instruções.

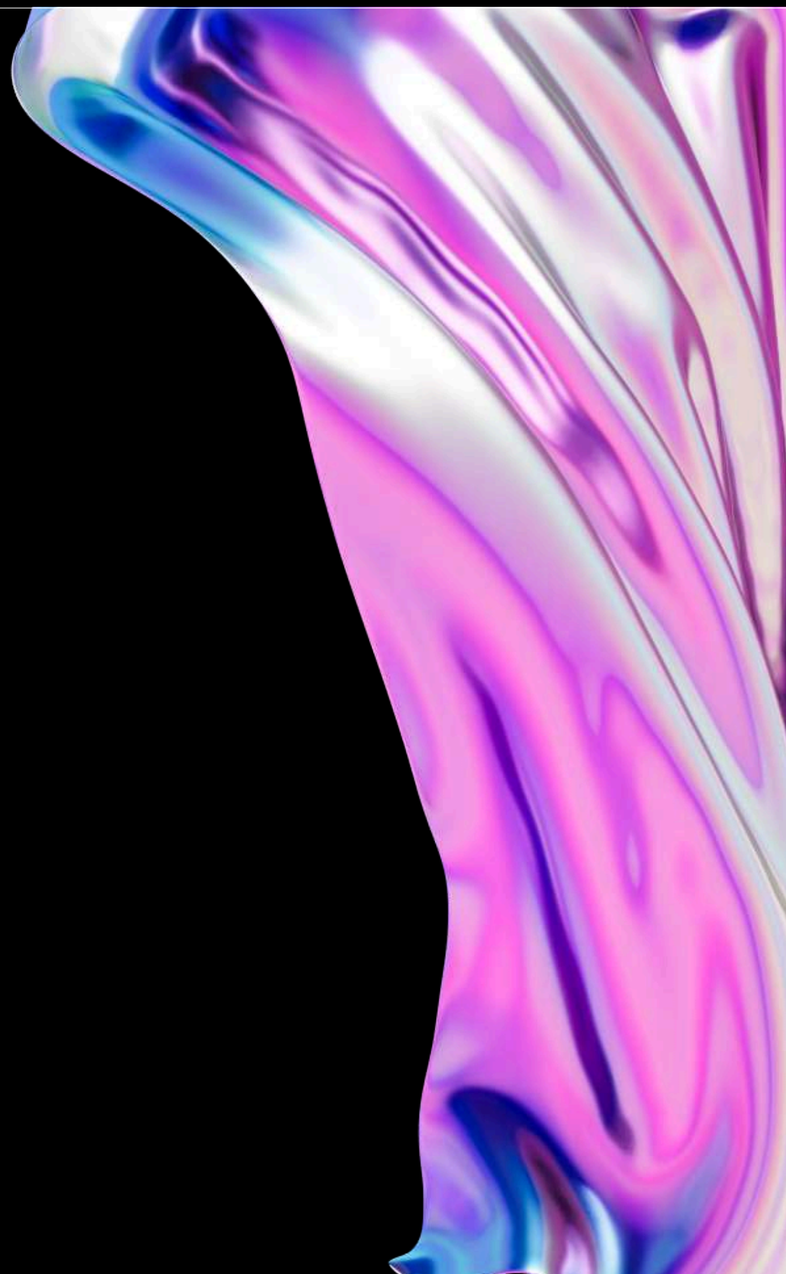
Verifique a instalação:

```
node -v
npm -v
```

Voltar para o índice

Instalação do Expo CLI

Expo CLI é uma ferramenta de linha de comando que facilita a criação e o gerenciamento de projetos React Native com Expo.



Linux, macOS e Windows

Instale o Expo CLI globalmente usando npm:

```
npm install -g expo-cli
```

Verifique a instalação:

```
expo --version
```

Voltar para o índice

Criação de novo projeto

Voltar para o índice

Continuando...

O comando expo start abrirá uma janela no navegador com o servidor de desenvolvimento Expo. Você pode escanear o QR code com o aplicativo Expo Go no seu dispositivo móvel para visualizar o projeto em tempo real.

Depois de configurar o ambiente e criar um novo projeto Expo, vamos configurar o projeto inicial e criar os componentes básicos. Vamos fazer isso em três partes:

1. Configuração Inicial do Projeto
2. Criando a Tela de Lista de Viagens
3. Criando a Tela de Detalhes da Viagem
4. Criando a Tela de Adição de Viagem

Voltar para o índice

Criando a Tela de Lista de Viagens

No terminal, dê o comando: `cd TravelDiary`

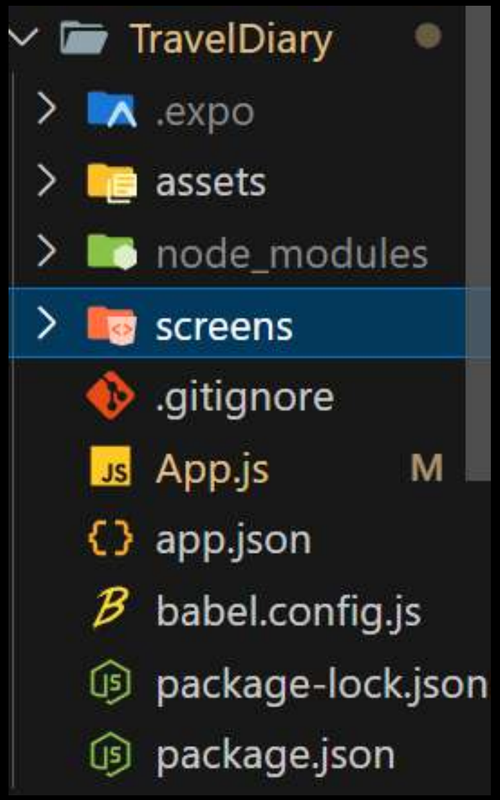
Em seguida, instalaremos as dependências necessárias para nosso projeto:

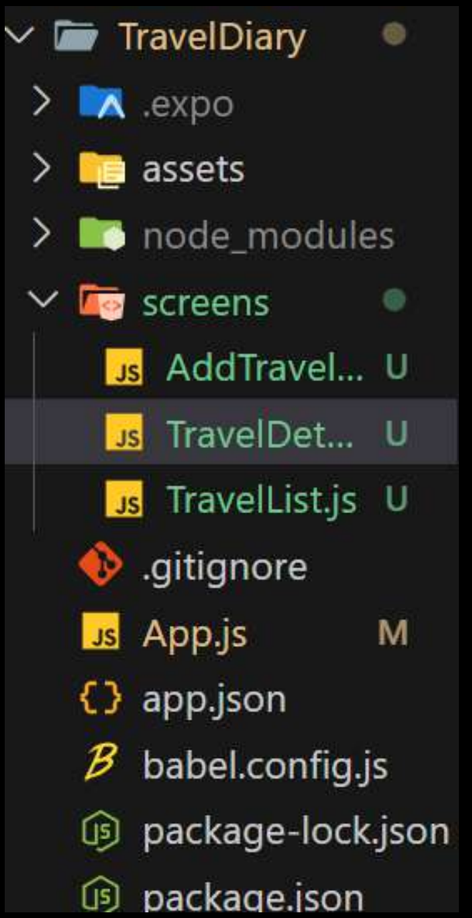
```
npm install @react-navigation/native @react-navigation/stack
npm install @react-native-async-storage/async-storage
expo install react-native-screens react-native-safe-area-context
expo install expo-image-picker
```

Criando as telas

Antes de qualquer outra coisa, dentro de TravelDiary, vamos criar uma pasta chamada screens

Dentro de screens, criaremos 3 arquivos: AddTravel.js, TravelDetails.js eTravelList.js





Criando a Tela de Adicionar Viagens

Explicação Geral

Este código define um componente funcional AddTravel que permite ao usuário adicionar uma nova viagem com destino, notas e fotos. O componente usa o useState para gerenciar o estado local e o AsyncStorage para armazenar os dados localmente no dispositivo. Também utiliza o ImagePicker da Expo para selecionar fotos da galeria do dispositivo.

Criando a Tela de Adicionar Viagens

```
// Importa os módulos necessários do React, React Native e outras bibliotecas
import React, { useState } from 'react';
import { View, TextInput, Button, Alert, Text, Image, ScrollView } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';
import * as ImagePicker from 'expo-image-picker';
```

Criando a Tela de Adicionar Viagens

```
1  // Define o componente AddTravel como uma função
2  export default function AddTravel({ navigation }) {
3    // Define o estado para destino, notas e fotos
4    const [destination, setDestination] = useState('');
5    const [notes, setNotes] = useState('');
6    const [photos, setPhotos] = useState([]);
```

Voltar para o índice

Criando a Tela de Adicionar Viagens

```
1  // Função para adicionar uma foto da galeria
2    const addPhoto = async () => {
3      // Abre a galeria de imagens para selecionar uma foto
4      let result = await ImagePicker.launchImageLibraryAsync({
5        mediaTypes: ImagePicker.MediaTypeOptions.Images,
6        allowsEditing: true,
7        aspect: [4, 3],
8        quality: 1,
9      });
```

Criando a Tela de Adicionar Viagens

```
// Verifica se o usuário selecionou uma foto ou cancelou a seleção
  if (!result.canceled) {
    const uri = result.assets[0].uri; // Obtém o URI da foto selecionada
    setPhotos([...photos, uri]); // Adiciona o URI da nova foto ao array de fotos
  } else {
    Alert.alert("Cancelled", "No photo selected"); // Mostra um alerta se a seleção for cancelada
  }
};
```

Quando falamos de URI, estamos nos referindo a um caminho para uma imagem armazenada localmente no dispositivo. Quando o usuário seleciona uma imagem da galeria usando ImagePicker, o resultado inclui um URI que aponta para a localização da imagem no dispositivo.

Criando a Tela de Adicionar Viagens

```
// Função para salvar as informações da viagem
 const saveTravel = async () => {
   // Verifica se os campos de destino e notas estão preenchidos
   if (!destination.trim() || !notes.trim()) {
     Alert.alert("Error", "Destination and Notes are required"); // Mostra um alerta se algum campo estiver vazio
     return;
   }

   // Cria um novo objeto de viagem com id, destino, notas e fotos
   const newTravel = {
     id: Date.now(), // Define um ID único com base no timestamp atual
     destination,
     notes,
     photos,
   };
```

Criando a Tela de Adicionar Viagens

```
try {
      // Tenta obter a lista de viagens armazenada
      const storedTravels = await AsyncStorage.getItem('travels');
      const travels = storedTravels ? JSON.parse(storedTravels) : []; // Analisa a lista existente ou cria uma nova lista vazia
      travels.push(newTravel); // Adiciona a nova viagem à lista
      await AsyncStorage.setItem('travels', JSON.stringify(travels)); // Armazena a lista atualizada
      Alert.alert("Success", "Travel saved successfully!"); // Mostra um alerta de sucesso
      navigation.navigate('TravelList', { refresh: true }); // Navega para a tela de lista de viagens e força uma atualização
    } catch (error) {
      console.error(error); // Registra o erro no console
      Alert.alert("Error", "An error occurred while saving the travel"); // Mostra um alerta se ocorrer um erro
    }
  };
```

Criando a Tela de Adicionar Viagens

```
// Renderiza o componente
  return (
    <ScrollView contentContainerStyle={{ padding: 20 }}>
      <TextInput
        placeholder="Destination" // Texto exibido quando o campo está vazio
        value={destination} // Valor do campo de destino
        onChangeText={setDestination} // Atualiza o estado de destino quando o texto é alterado
        style={{ marginBottom: 10, padding: 10, borderWidth: 1, borderColor: '#ccc' }} // Estilos para o campo de texto
      />
```

Voltar para o índice

Criando a Tela de Adicionar Viagens

```
<TextInput
        placeholder="Notes" // Texto exibido quando o campo está vazio
        value={notes} // Valor do campo de notas
        onChangeText={setNotes} // Atualiza o estado de notas quando o texto é alterado
        multiline // Permite múltiplas linhas
        style={{ marginBottom: 10, padding: 10, borderWidth: 1, borderColor: '#ccc', height: 100 }} // Estilos para o campo de texto
      />
```

Voltar para o índice

Criando a Tela de Adicionar Viagens

```
<Button title="Add Photo" onPress={addPhoto} /> // Botão para adicionar uma foto
      <Button title="Save Travel" onPress={saveTravel} style={{ marginTop: 10 }} /> // Botão para salvar a viagem
      {photos.length > 0 && (
        <View style={{ marginTop: 20 }}>
          <Text>Photos:</Text> // Texto que indica que as fotos estão listadas a seguir
          {photos.map((photo, index) => (
            <Image key={index} source={{ uri: photo }} style={{ width: 100, height: 100, marginTop: 10 }} /> // Exibe cada foto na lista
          ))}
        </View>
      )}
    </ScrollView>
  );
}
```

Criando a tela de Detalhes da Viagem

```
// Importa a biblioteca React
import React from 'react';
// Importa componentes nativos do React Native
import { View, Text, Image } from 'react-native';
```

```
// Define e exporta o componente funcional TravelDetails
export default function TravelDetails({ route }) {
    // Desestrutura o objeto route para obter os parâmetros passados, especificamente o
    // objeto travel
  const { travel } = route.params;

  return (
    <View> {/* Componente de contêiner para agrupar os elementos da tela */}
    {/* Exibe o destino da viagem */}
      <Text>Destination: {travel.destination}</Text>
```

Criando a tela de Detalhes da Viagem

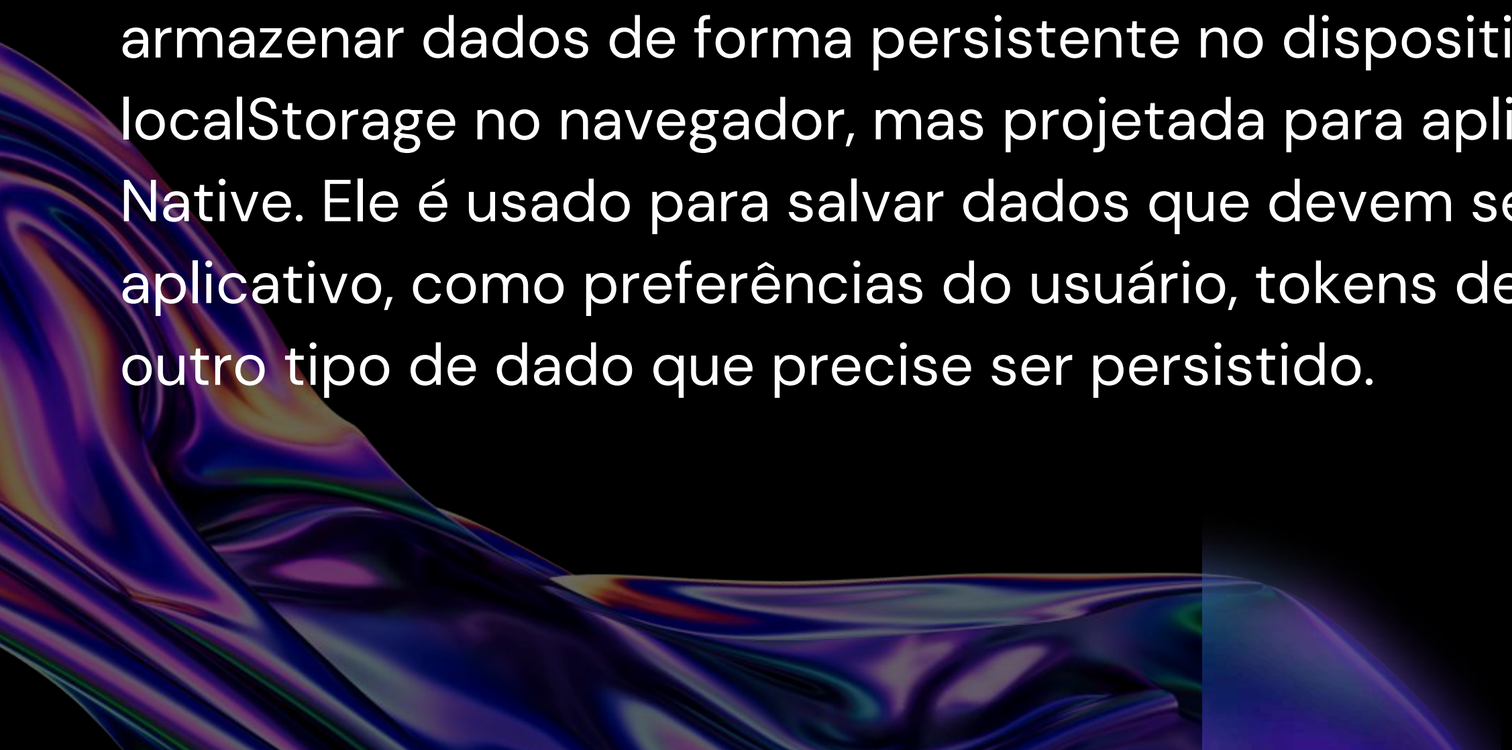
```
{/* Verifica se há fotos e as exibe */}
      {travel.photos && travel.photos.map((photo, index) => (
        <Image
          key={index} // Define uma chave única para cada imagem usando o índice do array
          source={{ uri: photo }} // Define a fonte da imagem usando o URI
          style={{ width: 100, height: 100 }} // Define o estilo da imagem (largura e altura)
        />
      ))}
    </View>
  );
}
```

Voltar para o índice

Tela de Lista de Viagens (TravelList.js)

O que é AsyncStorage?

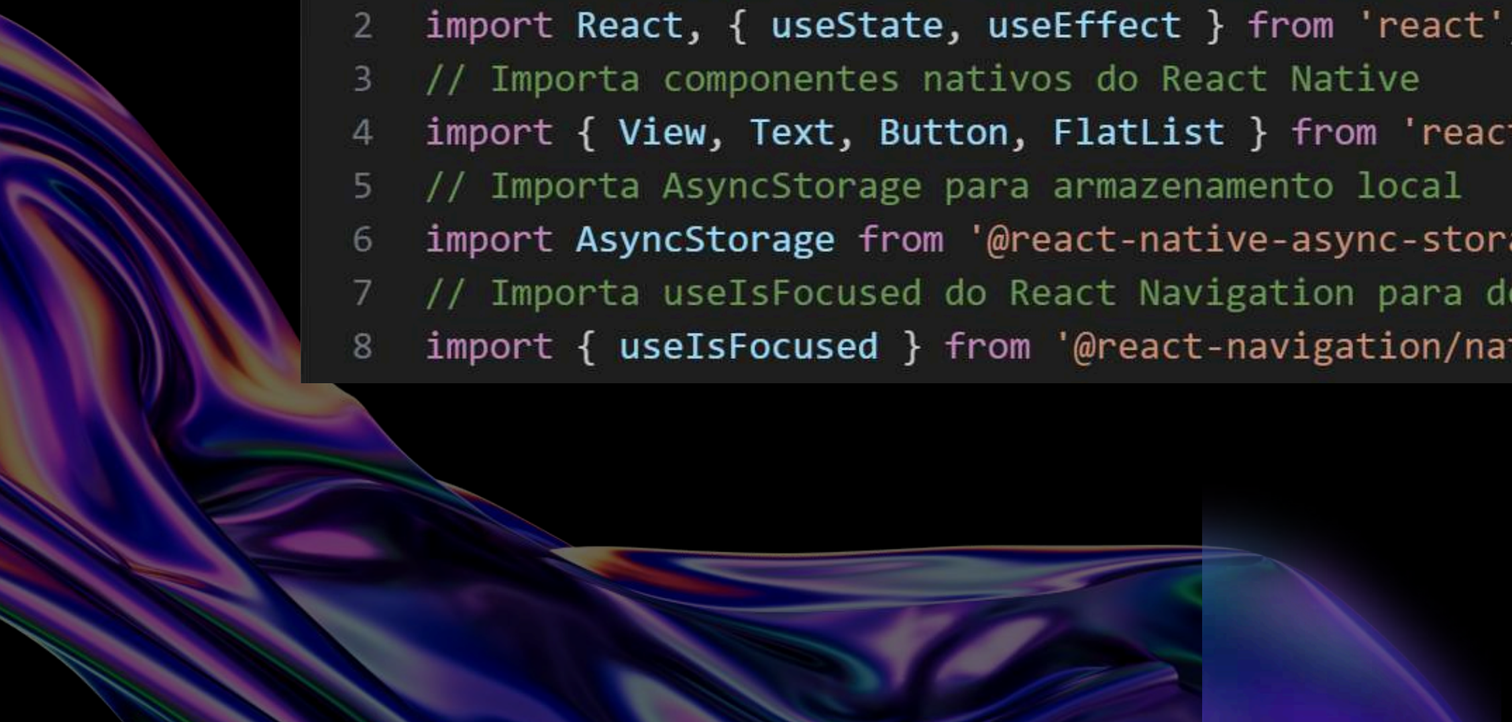
AsyncStorage é uma API de armazenamento local assincrona que permite armazenar dados de forma persistente no dispositivo do usuário, similar ao localStorage no navegador, mas projetada para aplicativos móveis com React Native. Ele é usado para salvar dados que devem ser mantidos entre as sessões do aplicativo, como preferências do usuário, tokens de autenticação, ou qualquer outro tipo de dado que precise ser persistido.



Voltar para o índice

Tela de Lista de Viagens (TravelList.js)

```
1  // Importa React e hooks useState e useEffect
2  import React, { useState, useEffect } from 'react';
3  // Importa componentes nativos do React Native
4  import { View, Text, Button, FlatList } from 'react-native';
5  // Importa AsyncStorage para armazenamento local
6  import AsyncStorage from '@react-native-async-storage/async-storage';
7  // Importa useIsFocused do React Navigation para detectar quando a tela está em foco
8  import { useIsFocused } from '@react-navigation/native';
```



Voltar para o índice

Tela de Lista de Viagens (TravelList.js)

Mas o que seria desestruturar o objeto navigation? Desestruturar o objeto navigation é uma forma de extrair propriedades de um objeto diretamente nas declarações de variável. Em JavaScript, a desestruturação permite acessar e usar essas propriedades sem precisar referenciar o objeto completo repetidamente.

```jsx
// Define e exporta o componente funcional TravelList
//e desestrutura o navigation
export default function TravelList({ navigation }) {
  // Estado para armazenar a lista de viagens
  const [travels, setTravels] = useState([]);
  // Hook para detectar quando a tela está em foco
  const isFocused = useIsFocused();
  // useEffect que carrega as viagens quando a tela está em foco
  useEffect(() => {
    // Função assíncrona para carregar as viagens do AsyncStorage
    const loadTravels = async () => {
      try {
        // Obtém as viagens armazenadas
        const storedTravels = await AsyncStorage.getItem('travels');
        // Se houver viagens armazenadas, atualiza o estado travels com elas
        if (storedTravels) setTravels(JSON.parse(storedTravels));
      } catch (error) {
        // Loga um erro no console se ocorrer
        console.error(error);
      }
    };

    // Chama a função loadTravels
    loadTravels();
  }, [isFocused]); // useEffect depende de isFocused,
  // então será executado sempre que a tela estiver em foco
```

Voltar para o índice

Tela de Lista de Viagens (TravelList.js)

```
return (
    // Componente de contêiner para a tela
    <View>
      {/* Botão para navegar para a tela de adicionar uma nova viagem */}
      <Button title="Add Travel" onPress={() => navigation.navigate('AddTravel')} />

      {/* FlatList para renderizar a lista de viagens */}
      <FlatList
        // Dados a serem renderizados
        data={travels}
        // Chave única para cada item na lista
        keyExtractor={(item) => item.id.toString()}
        // Função para renderizar cada item na lista
```

Voltar para o índice

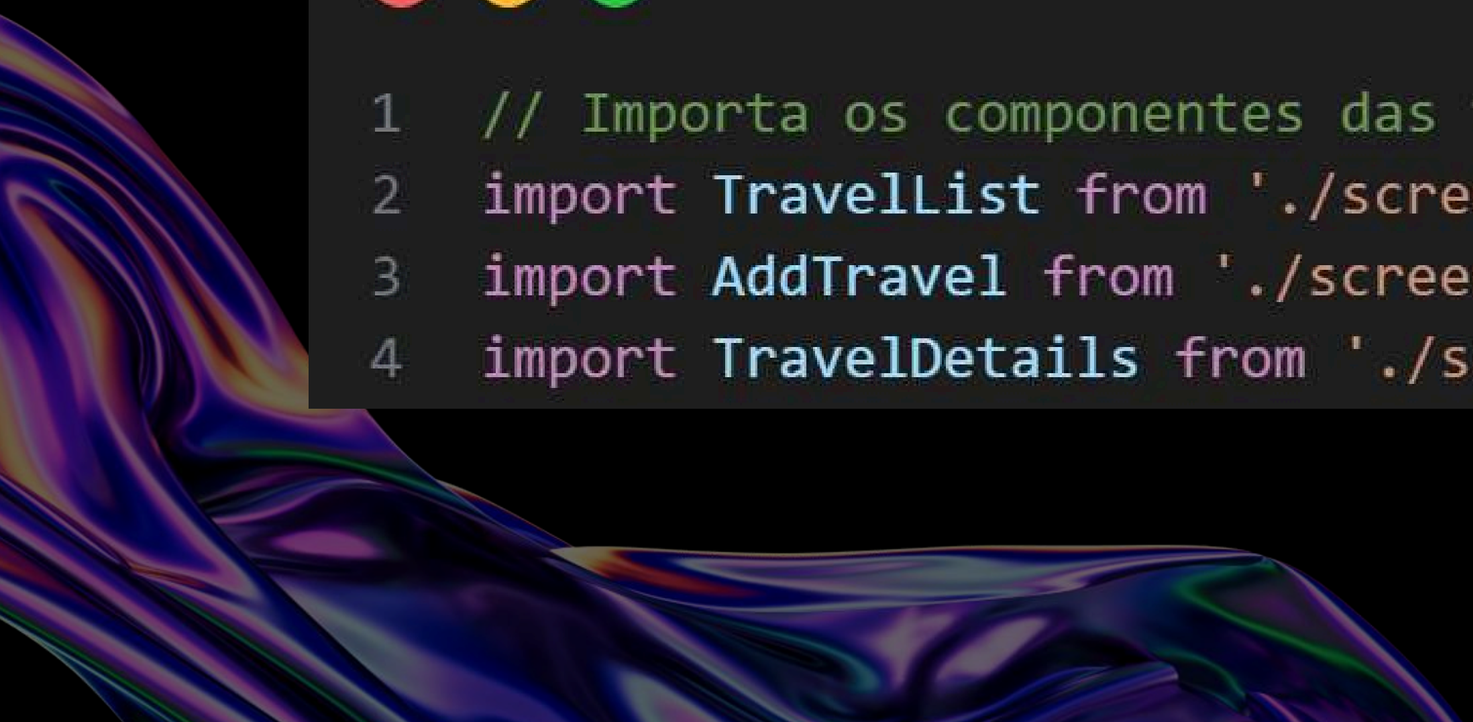
Tela de Lista de Viagens (TravelList.js)

```
        // Função para renderizar cada item na lista
        renderItem={({ item }) => (
          // Componente de contêiner para cada item da lista
          <View>
            {/* Exibe o destino da viagem */}
            <Text>{item.destination}</Text>
            {/* Botão para navegar para a tela de detalhes da viagem */}
            <Button title="Details" onPress={() => navigation.navigate('TravelDetails', { travel: item })} />
          </View>
        )}
      />
    </View>
  );
}
```

Voltar para o índice

Criando o arquivo App.js

```
// Importa os componentes das telas
import TravelList from './screens/TravelList';
import AddTravel from './screens/AddTravel';
import TravelDetails from './screens/TravelDetails';
```



Voltar para o índice

Criando o arquivo App.js

```
// Cria um objeto StackNavigator usando createStackNavigator
const Stack = createStackNavigator();

// Define o componente App como uma função
export default function App() {
  return (
    // NavigationContainer é o contêiner que gerencia o estado de navegação
    <NavigationContainer>
      {/* Define um Stack.Navigator para gerenciar a navegação entre telas */}
      <Stack.Navigator initialRouteName="TravelList">
        {/* Define a tela TravelList no stack de navegação */}
        <Stack.Screen name="TravelList" component={TravelList} />
        {/* Define a tela AddTravel no stack de navegação */}
        <Stack.Screen name="AddTravel" component={AddTravel} />
        {/* Define a tela TravelDetails no stack de navegação */}
        <Stack.Screen name="TravelDetails" component={TravelDetails} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Voltar para o índice

Renderizando nosso aplicativo

Para renderizar o aplicativo, vamos instalar no celular o App Expo Go.

No terminal, daremos o comando npx expo start, isso vai rodar o aplicativo e mostrar um QR code. Basta escanear com a câmera do celular, e será redirecionado para o expo go, mais precisamente para o aplicativo que criamos!