**Objetivos**

* Compreender o conceito de **API (Interface de Programação de Aplicações)** e sua importância.
* Aprender a fazer requisições **HTTP GET** em React Native.
* Utilizar a biblioteca **Axios** para simplificar as requisições.
* Exibir dados recebidos de uma API de forma dinâmica na interface.
* Gerenciar **estados de carregamento e erro** durante a requisição.
* **BÔNUS:** Discussão sobre useEffect para requisições de dados e tratamento de erros.

**Revisão Rápida**

* Como usamos props para passar dados entre componentes?
* Qual a função do StyleSheet e do Flexbox na estilização e layout?

**Conteúdo Teórico**

* **O que é uma API?**
  + Um conjunto de regras que permite a comunicação entre diferentes softwares.
  + Imagine-a como um "garçom" que leva seu pedido (requisição) para a "cozinha" (servidor) e traz a "comida" (resposta).
  + Exemplos comuns: APIs de redes sociais, mapas, dados climáticos.
* **Requisições HTTP (Método GET):**
  + **HTTP:** Protocolo de comunicação na web.
  + **GET:** O método mais comum para **solicitar dados** de um servidor.
  + Respostas da API: Geralmente vêm em formato **JSON**.
* **Axios vs. Fetch API:**
  + **Fetch API:** Nativa do JavaScript, boa para requisições simples.
  + **Axios:** Uma biblioteca popular que **simplifica e adiciona recursos** às requisições (interceptors, tratamento automático de JSON, melhor tratamento de erros). Será nosso foco.
* **Integração de Dados:**
  + Como transformar os dados JSON recebidos em algo que seu aplicativo possa usar.
  + Exibindo-os em uma lista (ex: FlatList).
* **Gerenciamento de Estado para Requisições:**
  + **isLoading:** Estado para mostrar um indicador de carregamento (ex: ActivityIndicator) enquanto os dados são buscados.
  + **error:** Estado para armazenar e exibir mensagens de erro em caso de falha na requisição.
  + **useEffect:** O Hook ideal para realizar requisições de dados ao montar/atualizar o componente.

**Prática Laboratorial**

**Passo 1: Instalar e Configurar Axios**

* **Ação:** Primeiro, instale a biblioteca Axios no seu projeto.
* **Comando:** No terminal, na raiz do seu projeto:

npm install axios

* **Explicação:**
  + Isso adiciona o Axios como uma dependência no seu package.json, tornando-o disponível para uso no seu código.

**Passo 2: Criar um Módulo de Serviço de API (services/api.js)**

* **Ação:** É uma boa prática centralizar a configuração do Axios em um arquivo separado para organizar seu código e facilitar futuras modificações.
* **Crie a pasta e o arquivo:** services/api.js
* **Conteúdo de services/api.js:**

// services/api.js

import axios from 'axios'; // <--- Importe o Axios

const api = axios.create({

baseURL: 'https://jsonplaceholder.typicode.com', // <--- URL base da API de exemplo

});

export default api; // <--- Exporte a instância configurada do Axios

* **Entendimento:**
  + axios.create(): Cria uma instância personalizada do Axios.
  + baseURL: Define a parte inicial da URL para todas as requisições feitas por essa instância. Isso evita repetição.
  + Usaremos o endpoint /posts dessa API, que retorna uma lista de dados de exemplo que podemos adaptar para "pontos turísticos".

**Passo 3: Buscar Dados da API com useEffect (ListaPontosTuristicos.js)**

* **Ação:** Agora, vamos modificar ListaPontosTuristicos.js para buscar dados da API quando a tela for carregada.
* **Mudança no screens/ListaPontosTuristicos.js:**

// screens/ListaPontosTuristicos.js

import React, { useState, useEffect } from 'react'; // <--- Importe useState e useEffect

import { View, Text, StyleSheet, ActivityIndicator, FlatList } from 'react-native'; // <--- Importe ActivityIndicator, FlatList

import PontoTuristicoCard from '../components/PontoTuristicoCard';

import api from '../services/api'; // <--- Importe a instância do Axios

const ListaPontosTuristicos = () => {

const [pontosTuristicos, setPontosTuristicos] = useState([]);

const [isLoading, setIsLoading] = useState(true); // <--- Estado para carregamento

const [error, setError] = useState(null); // <--- Estado para erros

useEffect(() => { // <--- Hook para executar a requisição ao montar o componente

const fetchPontosTuristicos = async () => {

try {

const response = await api.get('/posts'); // <--- Fazendo a requisição GET

// Adaptando os dados da API (jsonplaceholder) para nosso formato de ponto turístico

const dadosAdaptados = response.data.map(item => ({

id: String(item.id),

nome: item.title,

descricao: item.body,

imagem: `https://picsum.photos/id/${item.id % 100}/150/150`, // Imagem fictícia

// Adicionar aqui lat/lon se a API tivesse (será adicionado em aulas futuras)

}));

setPontosTuristicos(dadosAdaptados);

} catch (err) {

console.error("Erro ao buscar dados:", err);

setError("Não foi possível carregar os pontos turísticos."); // <--- Define a mensagem de erro

} finally {

setIsLoading(false); // <--- Finaliza o carregamento

}

};

fetchPontosTuristicos(); // <--- Chama a função de busca

}, []); // <--- Array de dependências vazio: executa apenas uma vez (ao montar)

if (isLoading) { // <--- Condição para mostrar o indicador de carregamento

return (

<View style={styles.loadingContainer}>

<ActivityIndicator size="large" color="#0000ff" />

<Text style={styles.loadingText}>Carregando pontos turísticos...</Text>

</View>

);

}

if (error) { // <--- Condição para mostrar a mensagem de erro

return (

<View style={styles.errorContainer}>

<Text style={styles.errorText}>{error}</Text>

</View>

);

}

return (

<View style={styles.container}>

<Text style={styles.mainTitle}>Pontos Turísticos</Text>

<FlatList // <--- Usando FlatList para exibir a lista

data={pontosTuristicos}

keyExtractor={(item) => item.id}

renderItem={({ item }) => (

<PontoTuristicoCard nome={item.nome} descricao={item.descricao} />

)}

/>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f5f5f5',

paddingTop: 50,

},

mainTitle: {

fontSize: 28,

fontWeight: 'bold',

textAlign: 'center',

marginBottom: 20,

color: '#333',

},

loadingContainer: { // Estilos para o carregamento

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#f5f5f5',

},

loadingText: {

marginTop: 10,

fontSize: 16,

color: '#666',

},

errorContainer: { // Estilos para o erro

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#ffe0e0',

},

errorText: {

fontSize: 16,

color: 'red',

textAlign: 'center',

marginHorizontal: 20,

}

// PontoTuristicoCard styles estão em components/PontoTuristicoCard.js

});

export default ListaPontosTuristicos;

* **Entendimento:**
  + useState([]): Inicializa a lista de pontos como um array vazio.
  + useState(true/false): Controla a visibilidade do ActivityIndicator.
  + useEffect(() => { ... }, []): Hook que executa a função de busca de dados **apenas uma vez** após a primeira renderização do componente (devido ao [] como segundo argumento).
  + api.get('/posts'): Realiza a requisição GET para o endpoint /posts da URL base configurada.
  + try...catch...finally: Bloco essencial para **tratamento de erros** (requisição falhou) e para garantir que isLoading seja definido como false ao final.
  + FlatList: Um componente otimizado para renderizar longas listas de dados, melhor que ScrollView para grandes volumes.
    - data: O array de dados a ser exibido.
    - keyExtractor: Função que retorna uma chave única para cada item (essencial para performance).
    - renderItem: Função que retorna o componente a ser renderizado para cada item.

**Desafio da Aula**

* No seu aplicativo "Guia Turístico":
  + Instale a biblioteca axios.
  + Crie o arquivo services/api.js e configure a instância do Axios com a baseURL fornecida.
  + No ListaPontosTuristicos.js:
    - Implemente a busca de dados usando useEffect e axios.
    - Utilize useState para pontosTuristicos, isLoading e error.
    - Adapte os dados recebidos da API jsonplaceholder.typicode.com/posts para o formato que seu PontoTuristicoCard espera (ex: title -> nome, body -> descricao).
    - Exiba um ActivityIndicator enquanto os dados estão sendo carregados.
    - Mostre uma mensagem de erro se a requisição falhar.
    - Use um componente **FlatList** para renderizar a lista de PontoTuristicoCards.
* Faça **commits no Git** com as alterações.

**Próximos Passos**

* Na próxima aula, vamos combinar a navegação com os dados dinâmicos, permitindo que o usuário clique em um item da lista e veja seus detalhes.