**Título**

Memória do App: Salvando Dados Localmente com AsyncStorage

**Objetivos**

* Compreender a necessidade e o conceito de **persistência de dados local**.
* Aprender a utilizar o **AsyncStorage** para armazenar e recuperar informações.
* Salvar e carregar dados como **strings JSON**.
* Integrar AsyncStorage para persistir dados simples (ex: configurações do usuário, lista de favoritos).
* **BÔNUS:** Boas práticas para armazenamento local e quando usar AsyncStorage vs. outras soluções.

**Revisão Rápida**

* Como passamos objetos completos entre telas usando navigation.navigate() e route.params?
* Qual a diferença entre isLoading e error no gerenciamento de estado de requisições API?

**Conteúdo Teórico**

* **Por que Persistir Dados Localmente?**
  + Aplicativos precisam lembrar de informações entre sessões (ex: login, configurações, itens favoritos).
  + Alternativas: armazenamento em nuvem (requer internet), banco de dados local (mais complexo).
  + AsyncStorage: Ideal para dados simples e não relacionais.
* **Introdução ao AsyncStorage:**
  + API de armazenamento de chave-valor (key-value storage) assíncrona.
  + Funciona como um pequeno "banco de dados" não relacional no dispositivo.
  + Dados são armazenados como **strings**.
  + Opera de forma assíncrona, ou seja, as operações não bloqueiam a interface do usuário.
* **Operações Básicas do AsyncStorage:**
  + setItem(key, value): Salva um par chave-valor. O value deve ser uma string.
  + getItem(key): Recupera o valor associado a uma chave. Retorna uma string ou null.
  + removeItem(key): Remove um par chave-valor.
  + clear(): Remove todos os dados do armazenamento (use com cuidado!).
* **Trabalhando com Objetos e Arrays:**
  + Como AsyncStorage só armazena strings, precisamos converter objetos/arrays para string (usando JSON.stringify()) antes de salvar.
  + Ao recuperar, convertemos de volta para objeto/array (usando JSON.parse()).
* **useEffect e AsyncStorage:**
  + useEffect é o hook ideal para carregar dados do AsyncStorage na montagem do componente.
  + Ele também é útil para salvar dados sempre que um estado relevante é alterado.

**Prática Laboratorial**

**Passo 1: Instalar AsyncStorage**

* **Ação:** O AsyncStorage é uma biblioteca separada que precisa ser instalada.
* **Comando:** No terminal, na raiz do seu projeto:
* npm install @react-native-async-storage/async-storage
* **Explicação:**
  + Este comando adiciona a biblioteca AsyncStorage ao seu projeto, permitindo que você a utilize para persistir dados.

**Passo 2: Criar um Contexto para Gerenciar Favoritos (context/FavoritesContext.js)**

* **Ação:** Vamos criar um **Contexto** para gerenciar os pontos turísticos favoritos. Isso é uma ótima maneira de encapsular a lógica de AsyncStorage e disponibilizar os favoritos para todo o app sem "prop drilling".
* **Crie a pasta e o arquivo:** context/FavoritesContext.js
* **Conteúdo de context/FavoritesContext.js:**
* // context/FavoritesContext.js
* import React, { createContext, useState, useContext, useEffect, useCallback } from 'react';
* import AsyncStorage from '@react-native-async-storage/async-storage'; // <--- Importe AsyncStorage
* const FavoritesContext = createContext(); // <--- Cria o Contexto
* const FAVORITES\_KEY = '@GuiaTuristico:favorites'; // <--- Chave para AsyncStorage
* export const FavoritesProvider = ({ children }) => {
* const [favoriteIds, setFavoriteIds] = useState([]); // <--- Armazena apenas os IDs dos favoritos
* const [isLoadingFavorites, setIsLoadingFavorites] = useState(true); // <--- Estado de carregamento
* // <--- Função para carregar favoritos do AsyncStorage
* const loadFavorites = useCallback(async () => {
* try {
* const storedFavorites = await AsyncStorage.getItem(FAVORITES\_KEY);
* if (storedFavorites !== null) {
* setFavoriteIds(JSON.parse(storedFavorites)); // <--- Converte de string para array
* }
* } catch (error) {
* console.error("Erro ao carregar favoritos do AsyncStorage:", error);
* } finally {
* setIsLoadingFavorites(false);
* }
* }, []);
* // <--- Função para salvar favoritos no AsyncStorage
* const saveFavorites = useCallback(async (ids) => {
* try {
* await AsyncStorage.setItem(FAVORITES\_KEY, JSON.stringify(ids)); // <--- Converte de array para string
* } catch (error) {
* console.error("Erro ao salvar favoritos no AsyncStorage:", error);
* }
* }, []);
* // <--- Efeito para carregar favoritos ao iniciar
* useEffect(() => {
* loadFavorites();
* }, [loadFavorites]); // Executa uma vez na montagem
* // <--- Efeito para salvar favoritos sempre que 'favoriteIds' mudar
* useEffect(() => {
* if (!isLoadingFavorites) { // Garante que não salve antes de carregar
* saveFavorites(favoriteIds);
* }
* }, [favoriteIds, isLoadingFavorites, saveFavorites]);
* // <--- Lógica para adicionar/remover favorito
* const toggleFavorite = (pontoId) => {
* setFavoriteIds(prevIds => {
* if (prevIds.includes(pontoId)) {
* return prevIds.filter(id => id !== pontoId); // Remove se já existe
* } else {
* return [...prevIds, pontoId]; // Adiciona se não existe
* }
* });
* };
* // <--- Lógica para verificar se é favorito
* const isFavorite = useCallback((pontoId) => {
* return favoriteIds.includes(pontoId);
* }, [favoriteIds]);
* const contextValue = {
* favoriteIds,
* isLoadingFavorites,
* toggleFavorite,
* isFavorite,
* };
* return (
* <FavoritesContext.Provider value={contextValue}>
* {children}
* </FavoritesContext.Provider>
* );
* };
* export const useFavorites = () => useContext(FavoritesContext); // <--- Hook customizado
* **Entendimento:**
  + **createContext():** Cria o "espaço" onde os dados e funções do contexto serão armazenados.
  + **FavoritesProvider:** Um componente que "fornece" o contexto para seus filhos. Envolvemos a parte do app que precisa acessar favoritos.
  + **FAVORITES\_KEY:** Uma chave única para identificar nossos dados no AsyncStorage.
  + **loadFavorites e saveFavorites:** Funções assíncronas para interagir com AsyncStorage, usando JSON.parse e JSON.stringify.
  + **useEffect:** Usado para carregar os favoritos quando o Provider é montado e para salvar os favoritos sempre que a lista favoriteIds é atualizada.
  + **useFavorites:** Um hook customizado que facilita o acesso aos valores do contexto em qualquer componente filho do Provider.

**Passo 3: Integrar o FavoritesProvider no App.js**

* **Ação:** Para que o contexto de favoritos esteja disponível para todo o seu aplicativo, você deve envolvê-lo com o FavoritesProvider no App.js.
* **Mudança em App.js:**
* // App.js
* import 'react-native-gesture-handler';
* import React from 'react';
* import { NavigationContainer } from '@react-navigation/native';
* import { createStackNavigator } from '@react-navigation/stack'; // Ou seu MainNavigator
* // <--- Importe o FavoritesProvider
* import { FavoritesProvider } from './context/FavoritesContext';
* // Importe suas telas
* import ListaPontosTuristicos from './screens/ListaPontosTuristicos';
* import DetalhesPontoTuristico from './screens/DetalhesPontoTuristico';
* // ... (outras telas se já estiverem no MainNavigator)
* const Stack = createStackNavigator(); // Ou use seu MainNavigator se já estiver complexo
* export default function App() {
* return (
* // <--- Envolva o NavigationContainer com o FavoritesProvider
* <FavoritesProvider>
* <NavigationContainer>
* <Stack.Navigator initialRouteName="ListaPontos">
* <Stack.Screen
* name="ListaPontos"
* component={ListaPontosTuristicos}
* options={{ title: 'Pontos Turísticos' }}
* />
* <Stack.Screen
* name="DetalhesPonto"
* component={DetalhesPontoTuristico}
* options={{ title: 'Detalhes do Ponto' }}
* />
* </Stack.Navigator>
* </NavigationContainer>
* </FavoritesProvider>
* );
* }
* **Entendimento:**
  + Ao envolver a NavigationContainer (ou o MainNavigator completo, se você já tiver um), garantimos que **todas as telas e componentes dentro da navegação** terão acesso ao contexto de favoritos.

**Passo 4: Usar o Contexto de Favoritos em PontoTuristicoCard e DetalhesPontoTuristico**

* **Ação:** Agora podemos usar o hook useFavorites para adicionar um botão de favoritar nos nossos cards e na tela de detalhes.
* **Mudança em components/PontoTuristicoCard.js (adicione um ícone de coração):**
* // components/PontoTuristicoCard.js
* import React from 'react';
* import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
* import { Ionicons } from '@expo/vector-icons'; // <--- Importe Ionicons
* import { useFavorites } from '../context/FavoritesContext'; // <--- Importe useFavorites
* const PontoTuristicoCard = ({ ponto, onPress }) => { // <--- Recebe o objeto 'ponto' completo
* const { isFavorite, toggleFavorite } = useFavorites(); // <--- Use o hook do contexto
* const handleToggleFavorite = () => {
* toggleFavorite(ponto.id); // <--- Usa a função do contexto com o ID do ponto
* };
* const favoriteIconName = isFavorite(ponto.id) ? 'heart' : 'heart-outline'; // <--- Lógica do ícone
* const favoriteIconColor = isFavorite(ponto.id) ? 'red' : 'gray';
* return (
* <TouchableOpacity onPress={onPress} style={styles.touchable}>
* <View style={styles.card}>
* <View style={styles.infoContainer}> {/\* <--- Novo container para info e favorito \*/}
* <Text style={styles.titulo}>{ponto.nome}</Text>
* <Text style={styles.descricao}>{ponto.descricao}</Text>
* </View>
* <TouchableOpacity onPress={handleToggleFavorite} style={styles.favoriteButton}>
* <Ionicons name={favoriteIconName} size={24} color={favoriteIconColor} />
* </TouchableOpacity>
* </View>
* </TouchableOpacity>
* );
* };
* const styles = StyleSheet.create({
* touchable: { width: '100%' },
* card: {
* flexDirection: 'row', // <--- Para alinhar info e botão lado a lado
* alignItems: 'center',
* backgroundColor: '#fff',
* padding: 15,
* marginVertical: 10,
* marginHorizontal: 20,
* borderRadius: 8,
* shadowColor: '#000', shadowOffset: { width: 0, height: 2 }, shadowOpacity: 0.1, shadowRadius: 3.84, elevation: 5,
* },
* infoContainer: { flex: 1, marginRight: 10 }, // <--- Para info ocupar o espaço restante
* titulo: { fontSize: 20, fontWeight: 'bold', marginBottom: 5, color: '#333' },
* descricao: { fontSize: 14, color: '#666' },
* favoriteButton: { padding: 5 }, // <--- Estilo do botão de favorito
* });
* export default PontoTuristicoCard;
* **Importante:** Você precisará instalar @expo/vector-icons se ainda não o fez: expo install @expo/vector-icons.
* **Adapte ListaPontosTuristicos.js para passar o objeto ponto completo para o card:**
* // screens/ListaPontosTuristicos.js (trecho do FlatList)
* <FlatList
* data={pontosTuristicos}
* keyExtractor={(item) => item.id}
* renderItem={({ item }) => (
* <PontoTuristicoCard
* ponto={item} // <--- Passa o objeto 'item' completo para o card
* onPress={() => handlePontoPress(item)}
* />
* )}
* />

**Passo 4 (Continuação): Usar o Contexto em DetalhesPontoTuristico.js**

* **Ação:** Adicione um botão de favoritar também na tela de detalhes.
* **Mudança em screens/DetalhesPontoTuristico.js:**
* // screens/DetalhesPontoTuristico.js
* import React from 'react';
* import { View, Text, StyleSheet, Button, ScrollView, TouchableOpacity } from 'react-native';
* import { useRoute, useNavigation } from '@react-navigation/native';
* import { Ionicons } from '@expo/vector-icons'; // <--- Importe Ionicons
* import { useFavorites } from '../context/FavoritesContext'; // <--- Importe useFavorites
* const DetalhesPontoTuristico = () => {
* const route = useRoute();
* const navigation = useNavigation();
* const { isFavorite, toggleFavorite } = useFavorites(); // <--- Use o hook do contexto
* const { pontoDetalhes } = route.params;
* if (!pontoDetalhes) {
* return (
* <View style={styles.container}>
* <Text style={styles.errorText}>Detalhes do ponto turístico não encontrados.</Text>
* <Button title="Voltar" onPress={() => navigation.goBack()} />
* </View>
* );
* }
* const handleToggleFavorite = () => {
* toggleFavorite(pontoDetalhes.id); // <--- Usa a função do contexto com o ID do ponto
* };
* const favoriteIconName = isFavorite(pontoDetalhes.id) ? 'heart' : 'heart-outline';
* const favoriteIconColor = isFavorite(pontoDetalhes.id) ? 'red' : 'gray';
* return (
* <ScrollView style={styles.scrollViewContainer}>
* <View style={styles.contentContainer}>
* <View style={styles.header}> {/\* <--- Novo container para título e favorito \*/}
* <Text style={styles.title}>{pontoDetalhes.nome}</Text>
* <TouchableOpacity onPress={handleToggleFavorite} style={styles.favoriteButton}>
* <Ionicons name={favoriteIconName} size={30} color={favoriteIconColor} />
* </TouchableOpacity>
* </View>
* <Text style={styles.detailText}>ID: {pontoDetalhes.id}</Text>
* <Text style={styles.descriptionText}>{pontoDetalhes.descricao}</Text>
* <Text style={styles.extraDetails}>Detalhes Completos: {pontoDetalhes.detalhesCompletos}</Text>
* <Button title="Voltar para a Lista" onPress={() => navigation.goBack()} />
* </View>
* </ScrollView>
* );
* };
* const styles = StyleSheet.create({
* scrollViewContainer: { flex: 1, backgroundColor: '#f5f5f5' },
* contentContainer: { padding: 20, alignItems: 'center' },
* header: { // <--- Estilo para alinhar título e botão de favorito
* flexDirection: 'row',
* justifyContent: 'space-between',
* alignItems: 'center',
* width: '100%', // Para ocupar toda a largura
* marginBottom: 15,
* },
* title: { fontSize: 28, fontWeight: 'bold', flexShrink: 1, marginRight: 10 }, // flexShrink para quebrar linha se o título for muito longo
* favoriteButton: { padding: 5 },
* detailText: { fontSize: 16, marginBottom: 10, color: '#666' },
* descriptionText: { fontSize: 16, lineHeight: 24, textAlign: 'justify', marginBottom: 20, color: '#444' },
* extraDetails: { fontSize: 14, fontStyle: 'italic', color: '#888', marginBottom: 30 },
* errorText: { fontSize: 18, color: 'red', textAlign: 'center', marginVertical: 50 }
* });
* export default DetalhesPontoTuristico;
* **Entendimento:**
  + Ambas as telas (PontoTuristicoCard e DetalhesPontoTuristico) agora usam o useFavorites para acessar e manipular a lista de favoritos, sem precisar passar props complexas por múltiplos níveis.
  + A interface do usuário é atualizada automaticamente quando um item é favoritato/desfavoritado.

**Desafio da Aula**

* No seu aplicativo "Guia Turístico":
  + Instale o AsyncStorage (@react-native-async-storage/async-storage).
  + Crie o arquivo context/FavoritesContext.js e implemente o FavoritesProvider e o useFavorites conforme os exemplos.
  + Adapte seu App.js para envolver a navegação principal (ou o MainNavigator se já tiver um) com o FavoritesProvider.
  + No PontoTuristicoCard.js, adicione um ícone de coração (usando @expo/vector-icons) que, ao ser clicado, adicione ou remova o ponto turístico da lista de favoritos, utilizando o toggleFavorite do contexto.
  + Na DetalhesPontoTuristico.js, adicione um ícone de coração similar que permita favoritar/desfavoritar o item diretamente da tela de detalhes.
  + **Opcional:** Crie uma nova tela FavoritosScreen.js (ainda sem navegação por aba) que apenas liste os IDs dos pontos favoritos, usando o favoriteIds do contexto.
* Faça **commits no Git** com as alterações.

**Próximos Passos**

* Na próxima aula, vamos implementar a navegação por abas (BottomTabNavigator), permitindo que o usuário alterne facilmente entre a lista de pontos, o mapa e a lista de favoritos.