Etec
Bento Quirino
Campinas

Guia Passo a Passo: Armazenamento Interno com AsyncStorage no React Native

Objetivo

Este guia fornece uma introdução ao uso do AsyncStorage no React Native, uma biblioteca que permite armazenar dados localmente no dispositivo. Esse recurso é útil para salvar informações de forma persistente entre sessões, como preferências de usuário, configurações ou dados simples.

Objetivos Gerais

1. Compreender o conceito de armazenamento interno em dispositivos móveis.
2. Aprender a configurar e utilizar o AsyncStorage no React Native.
3. Criar um aplicativo simples que utilize o AsyncStorage para salvar e recuperar dados.

Definição

AsyncStorage é uma biblioteca para React Native que fornece um sistema de armazenamento baseado em chave-valor de maneira assíncrona. Esse armazenamento é persistente e adequado para dados leves que precisam ser mantidos entre sessões. É ideal para salvar pequenas informações, como tokens de autenticação, preferências e estados simples de aplicativo.

Nota: A partir das versões mais recentes do React Native, o AsyncStorage foi movido para o pacote @react-native-async-storage/async-storage, devendo ser instalado separadamente.

Usos do AsyncStorage em Aplicativos Móveis

1. **Autenticação Persistente**:
   - Armazenar tokens de autenticação (como JWT) para manter o usuário logado entre sessões.
   - Evita que o usuário precise fazer login toda vez que abrir o aplicativo, melhorando a experiência de uso.
2. **Preferências do Usuário**:
   - Salvar configurações e preferências como temas (claro/escuro), configurações de idioma, notificações ou layout.
   - Isso permite que o usuário personalize o aplicativo, e essas escolhas sejam lembradas ao longo do uso.
3. **Dados de Formulário ou Estado Temporário**:
   - Manter dados não enviados, como respostas de formulários ou rascunhos de mensagens.

- Por exemplo, se o usuário começa a preencher um formulário e sai do aplicativo, ele pode retornar e encontrar o progresso salvo.

4. **Histórico de Pesquisa ou Navegação**:
   - Armazenar as consultas de pesquisa recentes, produtos visualizados, ou páginas acessadas.
   - Permite ao aplicativo mostrar um histórico personalizado ao usuário, como uma lista de pesquisas recentes.
5. **Cache de Dados de API**:
   - Salvar dados de API como listas de produtos, posts de blog ou mensagens.
   - Isso ajuda a reduzir chamadas de API e melhorar a velocidade de carregamento ao exibir dados recentes, mesmo quando o usuário está offline ou com internet lenta.

Esses são apenas alguns exemplos, mas o AsyncStorage pode ser útil sempre que você precisar armazenar dados leves que precisem ser persistentes entre sessões no aplicativo.

Exemplo Prático

Vamos criar um pequeno aplicativo de lista de tarefas ("To-do") que permite ao usuário adicionar e remover itens, e persiste esses dados usando AsyncStorage. Cada tarefa salva será recuperada automaticamente na próxima vez que o aplicativo for aberto.

Passo 1: Configuração Inicial

1. Crie um projeto React Native:

```
npx create-expo-app ToDoApp --template blank
cd ToDoApp
```

2. Instale o AsyncStorage:

```
npx expo install @react-native-async-storage/async-storage
```

3. Importe o AsyncStorage no seu código:

```
import AsyncStorage from '@react-native-async-storage/async-storage';
```

Passo 2: Estrutura do Aplicativo

Vamos desenvolver o aplicativo criando uma interface simples para adicionar e exibir as tarefas.

Código Completo

```
import React, { useState, useEffect } from 'react';
import { View, Text, TextInput,
         Button, FlatList, TouchableOpacity,
         StyleSheet, SafeAreaView } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

const App = () => {
  const [task, setTask] = useState('');
  const [tasks, setTasks] = useState([]);

  // Função para salvar tarefas no AsyncStorage
  const saveTasks = async (tasksArray) => {
    try {
      await AsyncStorage.setItem('tasks', JSON.stringify(tasksArray));
    } catch (error) {
      console.log('Erro ao salvar tarefas:', error);
    }
  };

  // Função para carregar tarefas ao iniciar o app
  const loadTasks = async () => {
    try {
      const storedTasks = await AsyncStorage.getItem('tasks');
      if (storedTasks !== null) {
        setTasks(JSON.parse(storedTasks));
      }
    } catch (error) {
      console.log('Erro ao carregar tarefas:', error);
    }
  };
```

```
// Adiciona uma nova tarefa
const addTask = () => {
  if (task.trim() !== '') {
    const newTasks = [...tasks, task];
    setTasks(newTasks);
    saveTasks(newTasks); // Salva a lista atualizada
    setTask(''); // Limpa o campo de entrada
  }
};

// Remove uma tarefa pelo índice
const removeTask = (index) => {
  const newTasks = tasks.filter((_, i) => i !== index);
  setTasks(newTasks);
  saveTasks(newTasks); // Salva a lista atualizada
};

// Carrega as tarefas ao iniciar o aplicativo
useEffect(() => {
  loadTasks();
}, []);

```

```
1    return (
2      <View style={styles.container}>
3        <Text style={styles.title}>Lista de Tarefas</Text>
4        <TextInput
5          style={styles.input}
6          placeholder="Digite uma nova tarefa"
7          value={task}
8          onChangeText={(text) => setTask(text)}
9        />
10       <Button title="Adicionar Tarefa" onPress={addTask} />
11       <FlatList
12         data={tasks}
13         keyExtractor={(item, index) => index.toString()}
14         renderItem={({ item, index }) => (
15           <View style={styles.taskContainer}>
16             <Text style={styles.taskText}>{item}</Text>
17             <TouchableOpacity onPress={() => removeTask(index)}>
18               <Text style={styles.deleteText}>Excluir</Text>
19             </TouchableOpacity>
20           </View>
21         )}
22       />
23     </View >
24   );
25 };
26
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    backgroundColor: '#f5f5f5',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginTop: 20,
    marginBottom: 20,
  },
  input: {
    height: 40,
    borderColor: '#cccccc',
    borderWidth: 1,
    marginBottom: 10,
    paddingHorizontal: 10,
  },
  taskContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    paddingVertical: 10,
    borderBottomWidth: 1,
    borderBottomColor: '#ddd',
  },
  taskText: {
    fontSize: 18,
  },
  deleteText: {
    color: 'red',
    fontWeight: 'bold',
  },
});

export default App;
```

Passo 3: Explicação do Código

1. **Carregamento de Tarefas**: No `useEffect`, chamamos `loadTasks()` para recuperar as tarefas salvas no `AsyncStorage` ao iniciar o aplicativo.
2. **Adicionar e Salvar Tarefas**: A função `addTask()` adiciona a tarefa digitada e atualiza a lista de tarefas. Em seguida, a função `saveTasks()` armazena a lista atualizada no `AsyncStorage`.
3. **Remover e Atualizar Tarefas**: A função `removeTask()` remove a tarefa da lista pelo índice, atualiza o estado e salva a nova lista.
4. **Interface**: Utilizamos um `TextInput` para digitar novas tarefas, um botão para adicionar e um `FlatList` para exibir cada tarefa. Cada tarefa inclui um botão "Excluir" para removê-la.

Passo 4: Teste do Aplicativo

- **Adicionar Tarefa**: Digite uma tarefa no campo de entrada e pressione "Adicionar Tarefa". A tarefa será salva automaticamente no armazenamento interno.
- **Excluir Tarefa**: Pressione "Excluir" ao lado de uma tarefa para removê-la.
- **Persistência de Dados**: Feche e abra o aplicativo. As tarefas devem permanecer salvas.

Conclusão

Com este aplicativo, você aprendeu a usar o `AsyncStorage` para armazenar e recuperar dados no React Native. Esse recurso é útil para manter informações leves entre sessões, garantindo que o usuário possa continuar de onde parou.