Códigos de Alta performance MOBILE

Prof. Maurício Aldenor

040601707@prof.unama.br

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

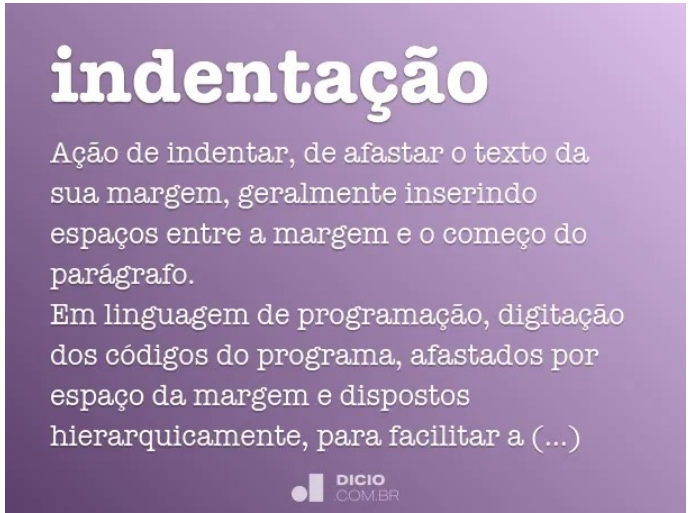
Qual boas práticas você usa ao desenvolver um aplicativo móvel?

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

1) Organização do código e Reutilização de Código

Objetivo: Estruturar o projeto para facilitar a leitura e manutenção, além Usar componentes reutilizáveis para evitar duplicidade.

Exemplo: Indentação, pastas para componentes, telas, serviços, assets, Criar botões, cabeçalhos e outros elementos comuns como componentes separados.



Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

1) Organização do código e Reutilização de Código

```
project-root/
├── assets/                 # Imagens, ícones e outros recursos de mídia
│   ├── images/
│   └── icons/
├── src/                    # Código fonte principal do aplicativo
│   ├── components/         # Componentes reutilizáveis
│   │   ├── Button.js
│   │   ├── Header.js
│   │   └── Card.js
│   ├── screens/            # Páginas principais da aplicação
│   │   ├── HomeScreen.js
│   │   ├── LoginScreen.js
│   │   └── ProfileScreen.js
│   ├── navigation/         # Arquivos de navegação
│   │   └── AppNavigator.js
│   ├── services/           # Funções e módulos de integração com APIs
│   │   └── apiService.js
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

1) Organização do código e Reutilização de Código

```jsx
// src/components/Button.js
import React from 'react';
import { TouchableOpacity, Text, StyleSheet } from 'react-native';

const Button = ({ title, onPress, style }) => {
  return (
    <TouchableOpacity style={[styles.button, style]} onPress={onPress}>
      <Text style={styles.buttonText}>{title}</Text>
    </TouchableOpacity>
  );
};
```

```jsx
// src/screens/HomeScreen.js
import React from 'react';
import { View, Text } from 'react-native';
import Button from '../components/Button';

export default function HomeScreen() {
  return (
    <View>
      <Text>Bem-vindo ao App</Text>
      <Button title="Acessar Perfil" onPress={() => console.log("Navegar para perfil")}
    </View>
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

1) Organização do código e Reutilização de Código

Vantagens da Organização

1. **Manutenção Facilitada**: Localizar arquivos e componentes específicos é mais fácil, e isso agiliza a manutenção.
2. **Modularidade**: Cada parte do código tem uma responsabilidade bem definida, facilitando a leitura.
3. **Reutilização**: Componentes e serviços reutilizáveis ajudam a reduzir a duplicidade de código.
4. **Escalabilidade**: A estrutura permite que o projeto cresça sem perder a organização, uma vantagem em projetos grandes.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

2) Nomenclatura Consistente

Objetivo: Usar nomes descritivos para variáveis e funções.

- **Exemplo:** usar getUserData ao invés de getData, usar CamelCase, usar PascalCase, evitar abreviações, constantes em UPPER_CASE.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

2) Nomenclatura Consistente

```
const d = "Nome de Usuário";
const msg = "Bem-vindo ao aplicativo!";
const num = 5;

function fn(x) {
  return x * num;
}
```

```
const userName = "Nome de Usuário";      // Nome significativo e em camelCase
const welcomeMessage = "Bem-vindo ao aplicativo!";  // Nome descritivo
const maxAttempts = 5;  // Indica o propósito da variável

function multiplyByMaxAttempts(value) {  // Nome que descreve a ação da função
  return value * maxAttempts;
}
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

2) Nomenclatura Consistente

```
const apiurl = "https://api.exemplo.com";
const MaxTries = 3;
```

```
const API_URL = "https://api.exemplo.com";  // Nome de constante em UPPER_SNAKE_CASE
const MAX_RETRY_ATTEMPTS = 3;
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

2) Nomenclatura Consistente

Vantagens

É importante que nomes de variáveis, funções, componentes e outros elementos sejam descritivos e sigam convenções bem definidas. Isso melhora a legibilidade e a compreensão do código para qualquer desenvolvedor que precise fazer manutenção ou expandir o projeto.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

3) Gerenciamento de Estado

Objetivo: Usar Context API ou Redux para dados globais.

O gerenciamento de estado é essencial em aplicativos React Native para manter dados sincronizados e atualizados entre diferentes componentes e telas. O React Native oferece várias formas de gerenciar o estado, sendo o **useState** e **useReducer** do React bons para componentes menores, e o **Context API** e bibliotecas como **Redux** úteis para gerenciar estados globais de forma eficiente.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

3) Gerenciamento de Estado

useState gerencia o estado local **count** dentro do componente. Esse estado é restrito ao componente **CounterScreen** e é atualizado quando o botão é pressionado (LOCAL)

```
import React, { useState } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';

export default function CounterScreen() {
  const [count, setCount] = useState(0); // Estado local com useState

  return (
    <View style={styles.container}>
      <Text>Contador: {count}</Text>
      <Button title="Incrementar" onPress={() => setCount(count + 1)} />
    </View>
  );
}
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

3) Gerenciamento de Estado

o AuthContext fornece um estado de autenticação GLOBAL acessível em qualquer componente que use AuthContext, sendo útil para estados que precisam ser compartilhados entre diferentes partes do app.

```jsx
// AuthContext.js
import React, { createContext, useState } from 'react';

export const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => setIsAuthenticated(true);
  const logout = () => setIsAuthenticated(false);

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
```

```jsx
// App.js
import React from 'react';
import { AuthProvider } from './src/AuthContext';
import HomeScreen from './src/screens/HomeScreen';

export default function App() {
  return (
    <AuthProvider>
      <HomeScreen />
    </AuthProvider>
  );
}
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

4) Separação de Lógica e UI

Separar a lógica de negócios da UI (interface do usuário) é uma prática essencial em desenvolvimento de software, pois torna o código mais modular, fácil de entender e de manter. Essa separação pode ser feita organizando o código em **componentes de apresentação** e **componentes de contêiner**.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

4) Separação de Lógica e UI

O UserList apenas recebe e exibe uma lista de usuários (users). Ele não sabe como os dados são obtidos, pois sua única responsabilidade é a apresentação.

```
export default function UserList({ users }) {
  return (
    <View style={styles.container}>
      <FlatList
        data={users}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View style={styles.userItem}>
            <Text style={styles.userName}>{item.name}</Text>
          </View>
        )}
      />
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

4) Separação de Lógica e UI

UserListContainer é responsável por obter os dados dos usuários e gerenciar o estado. Quando o componente é montado, ele simula uma chamada de API para buscar dados de usuários e os passa como props para o UserList

```
import UserList from './UserList';

export default function UserListContainer() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    // Simulação de uma chamada de API para buscar os usuários
    fetchUsers();
  }, []);

  const fetchUsers = async () => {
    // Simulação de dados de usuários
    const mockUsers = [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' },
      { id: 3, name: 'Charlie' },
    ];
    setUsers(mockUsers);
```

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

4) Separação de Lógica e UI

Características:

-Componentes de Apresentação: Apenas exibem dados e interagem com o usuário. Eles não contêm lógica de negócios.

-Componentes de Lógica (Contêineres): Gerenciam estados e dados, além de passar esses dados para os componentes de apresentação. Eles geralmente interagem com APIs ou outros serviços externos.

-Serviços Externos: Funções ou módulos separados para lógica complexa (como chamadas de API), mantêm o código mais limpo e reutilizável.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

5) Melhoria de Desempenho com Memoização

É uma técnica que ajuda a melhorar o desempenho do aplicativo guardando o valor de funções para que possam ser reutilizados e evitando renderizações desnecessárias.

Melhores Práticas de Desenvolvimento de Software para Dispositivos Mobile

5) Melhoria de Desempenho com Memoização

o ItemList é memoizado com **React.memo**, de forma que ele só re-renderiza se os items mudarem. O botão "Incrementar Contador" aumenta o contador no componente pai, mas ItemList não é re-renderizado quando o contador muda, pois os items não foram alterados.

```jsx
// ItemList.js
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const ItemList = React.memo(({ items }) => {
  console.log("Renderizou ItemList");
  return (
    <View style={styles.container}>
      {items.map((item, index) => (
        <Text key={index} style={styles.itemText}>{item}</Text>
      ))}
    </View>
  );
});
```

```jsx
// ParentComponent.js
import React, { useState } from 'react';
import { View, Button, Text } from 'react-native';
import ItemList from './ItemList';

export default function ParentComponent() {
  const [counter, setCounter] = useState(0);
  const items = ["Item 1", "Item 2", "Item 3"];

  return (
    <View>
      <Text>Contador: {counter}</Text>
      <Button title="Incrementar Contador" onPress={() => setCounter(counter + 1)} />
      <ItemList items={items} />
```

DÚVIDAS?

Banco de Dados para Mobile

Exercício 01

1) Criar uma aplicação simples que exiba uma lista de tarefas, onde seja possível adicionar uma nova tarefa e marcar uma tarefa como concluída. Você deve usar um gerenciamento de estado adequado, uma estrutura de projeto organizada e otimizar o desempenho com memoização.

Banco de Dados para Mobile

Exercício 01

1) Requisitos

- Estruturar o projeto para facilitar a leitura e manutenção: Organizar o código em pastas de forma clara.

- Usar nomenclatura consistente: Nomear componentes e funções de forma que o propósito deles fique claro.
- Gerenciar o estado corretamente: Utilizar useState e useReducer para o gerenciamento de estado.
- Separação de lógica e UI: Criar componentes separados para lógica e UI.
- Memoização para otimização: Usar **React.memo e/ou useCallback** para evitar renderizações desnecessárias.

Banco de Dados para Mobile

Exercício 01

Estrutura Básica

Banco de Dados para Mobile

Exercício 01

Resultado Esperado

Ao executar o aplicativo:

•O usuário pode adicionar novas tarefas ao inserir texto e clicar em "Adicionar Tarefa".

•A lista exibe cada tarefa adicionada.

•Ao tocar em uma tarefa, ela é marcada como concluída, e o texto é riscado.

•Alterações de estado e renderizações ocorrem de forma otimizada, aplicando memoização.