1. **Atualização Tecnológica:**

   Para me manter atualizado com novas tecnologias, utilizo uma combinação de métodos:

   - Documentação oficial: Sempre começo pela documentação oficial das tecnologias que uso, como React Native, React, JavaScript, etc.
   - Blogs técnicos: Leio regularmente blogs como Medium;
   - YouTube: Canais como Fireship, Traversy Media, e Ben Awad oferecem conteúdo atualizado e prático.
   - Cursos online: Plataformas como Udemy para aprendizado mais estruturado.
   - GitHub: Acompanho repositórios relevantes para ver novas features e discussões.
   - Linkedin: Gosto de utliza-lo para me conectar com desenvolvedores com presença nas comunidades e referência no mercado acompanhando seus desenvolvimentos.

2. **Funções no JavaScript:**

   Tanto function quanto arrow function têm seus lugares no desenvolvimento React.

   Function:

```
function MyComponent(props) {
  this.state = { count: 0 };
  this.handleClick = function() {
    this.setState({ count: this.state.count + 1 });
  };

  return <button onClick={this.handleClick}>Click me</button>;
}
```

   Arrow Function:

```
const MyComponent = (props) => {
  const [count, setCount] = useState(0);
```

```
 const handleClick = () => {
  setCount(count + 1);
 };

 return <button onClick={handleClick}>Click me</button>;
};
```

Vantagens da function:

- Mais clara para componentes complexos com muitos métodos.
- Permite o uso de 'this' para referir-se à instância do componente.

Vantagens da arrow function:

- Sintaxe mais concisa, especialmente para componentes simples.
- Não redefine o 'this', tornando-a útil para callbacks.
- Ótima para componentes funcionais com hooks.

Minha opinião é usar arrow functions para componentes funcionais simples e hooks, e funções regulares para componentes de classe ou funções mais complexas que precisam de "this"ou hoisting.

3. **Solução de Problemas:**

Um desafio que enfrentei foi implementar um sistema de chat bot.

Foi desenvolvido com React Native e integrando com o chat bot da AWS, foi a minha primeira implementação integrando esse serviço da AWS, o maior problema foi fazer a conversa entre as duas pontas acontecerem.

Na plataforma do AWS era feita configuração das perguntas e possíveis respostas, mas ele também aprendia conforme a conversa avança, Usei Redux para gerenciar o estado global do chat.

Outro desafio foi desenvolver um aplicativo para uma professora para colocar seus conteúdos onde seus alunos pudessem acessar. Utilizei Expo EAS, React Native, Tailwind e o Payload para parte de dados.

Um dos grandes desafios foi realizar a subida do aplicativo paras lojas Google e Apple, onde utilizei o Expo EAS para parte de build e subir nas lojas.

4. **React Native e Context API/Hooks:**

Problema: Gerenciamento de tema (claro/escuro) em um aplicativo.

Solução:

```
// ThemeContext.js
import React, { createContext, useState, useContext } from 'react';

const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
 const [isDarkMode, setIsDarkMode] = useState(false);

 const toggleTheme = () => setIsDarkMode(!isDarkMode);

 return (
  <ThemeContext.Provider value={{ isDarkMode, toggleTheme }}>
   {children}
  </ThemeContext.Provider>
 );
};

export const useTheme = () => useContext(ThemeContext);
```

```
// App.js
import { ThemeProvider } from './ThemeContext';
```

```
const App = () => (
  <ThemeProvider>
    <MainApp />
  </ThemeProvider>
);

// Component.js
import { useTheme } from './ThemeContext';

const MyComponent = () => {
  const { isDarkMode, toggleTheme } = useTheme();

  return (
    <View style={{ backgroundColor: isDarkMode ? 'black' : 'white' }}>
      <Button onPress={toggleTheme} title="Toggle Theme" />
    </View>
  );
};
```

Esta solução permitiu um gerenciamento de tema eficiente e fácil de usar em toda a aplicação, melhorando a experiência do usuário e facilitando a manutenção do código.

5. **Roteamento:**

   Para configurar o roteamento em React Native, eu usaria o React Navigation. É a biblioteca mais popular e bem mantida para navegação em React Native.

   Exemplo de configuração:

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

function App() {
 return (
  <NavigationContainer>
   <Stack.Navigator>
    <Stack.Screen name="Home" component={HomeScreen} />
    <Stack.Screen name="Details" component={DetailsScreen} />
   </Stack.Navigator>
  </NavigationContainer>
 );
}
```

   Escolhi React Navigation porque:

   - Tem uma API intuitiva e fácil de usar.
   - Oferece vários tipos de navegação (stack, tab).
   - Tem bom desempenho e é amplamente adotado pela comunidade.
   - Fornece uma experiência de navegação nativa em iOS e Android.

- Tem boa documentação e suporte da comunidade.

  Mas também sei que tem o Expo Router a qual já estudei e achei muito bom, mas o React Navigation foi o que mais usei.

6. **Depuração de Bugs:**
   Para identificar e resolver um problema de travamento em um dispositivo específico:

   - Reproduzir o problema: Tentar replicar o bug no dispositivo específico e em outros para isolar se é um problema do dispositivo ou do app.
   - Logs: Adicionar logs detalhados em pontos críticos do código para entender o fluxo de execução.
   - React Native Debugger: Usar esta ferramenta para inspecionar o estado da aplicação e as props dos componentes.
   - Crashlytics: Se o app estiver em produção, usar uma ferramenta como Crashlytics para obter relatórios detalhados de crashes.
   - Metro Bundler: Limpar o cache do Metro bundler e reconstruir o app.
   - Análise de código: Revisar o código em busca de memory leaks ou uso excessivo de recursos.
   - Testes em diferentes versões: Testar o app em diferentes versões do sistema operacional do dispositivo;
   - Profiling: Usar ferramentas de profiling nativas (Xcode Instruments para iOS, Android Profiler para Android) para identificar problemas de performance.
   - Flipper: Para conseguir visualizar as requests para APIs.

   Após identificar a causa, implementaria a correção, testaria extensivamente e monitoraria para garantir que o problema foi resolvido.

   Essas práticas ajudam a identificar e resolver problemas de forma sistemática e eficiente.