Spécification Technique pour le Contexte `SocketContext`

1. Introduction

Le fichier `socketContext.tsx` configure et fournit un contexte pour la gestion de la connexion WebSocket à l'aide de la bibliothèque `socket.io-client`. Ce contexte permet aux composants React d'accéder et d'utiliser facilement la connexion WebSocket dans l'ensemble de l'application.

2. Objectifs

- Initialiser et configurer la connexion WebSocket avec le serveur.
- Fournir un contexte global pour la connexion WebSocket.
- Permettre aux composants React d'accéder et d'utiliser la connexion WebSocket via un hook personnalisé.

3. Dépendances

- `react`
- `socket.io-client`

4. Interface

4.1. Contexte

- `SocketContext`: Un contexte React créé pour stocker la connexion WebSocket.

4.2. Hooks

- `useSocket`: Un hook personnalisé pour accéder à la connexion WebSocket depuis n'importe quel composant.

4.3. Fournisseur de Contexte

- `SocketProvider`: Un fournisseur de contexte qui encapsule les composants enfants et leur fournit la connexion WebSocket via le contexte `SocketContext`.

5. Fonctionnalités

5.1. Initialisation du WebSocket

Le WebSocket est initialisé à l'aide de `socket.io-client` avec une connexion à `ws://localhost:8001` et l'utilisation du transport `websocket`.

```
const socket = io("ws://localhost:8001", { transports: ['websocket']
});
```

5.2. Création du Contexte

Un contexte React est créé pour stocker la connexion WebSocket.

```
const SocketContext = createContext<Socket | null>(null);
```

5.3. Hook Personnalisé

Le hook `useSocket` permet d'accéder à la connexion WebSocket depuis n'importe quel composant.

```
export const useSocket = () => {
  return useContext(SocketContext);
};
```

5.4. Fournisseur de Contexte

Le fournisseur `SocketProvider` encapsule les composants enfants et leur fournit la connexion WebSocket via le contexte `SocketContext`.

```
export const SocketProvider: React.FC<{ children: React.ReactNode }>
= ({ children }) => {
  return (
    <SocketContext.Provider value={socket}>
      {children}
    </SocketContext.Provider>
  );
};
```

6. Structure du Composant

```
import React, { createContext, useContext } from 'react';
import { io, Socket } from 'socket.io-client';

const socket = io("ws://localhost:8001", { transports: ['websocket']
});

const SocketContext = createContext<Socket | null>(null);

export const useSocket = () => {
  return useContext(SocketContext);
};

export const SocketProvider: React.FC<{ children: React.ReactNode }>
= ({ children }) => {
  return (
    <SocketContext.Provider value={socket}>
      {children}
    </SocketContext.Provider>
  );
};
```

7. Utilisation

Pour utiliser le contexte dans un composant, il faut encapsuler les composants dans `SocketProvider` et utiliser le hook `useSocket` pour accéder à la connexion WebSocket.

Exemple d'Utilisation

```
// App.tsx
import React from 'react';
import { SocketProvider } from './context/socketContext';
import { TotalAmountProvider } from './context/totalAmountContext';
import MainApp from './MainApp';

export default () => (
  <SocketProvider>
    <TotalAmountProvider>
      <MainApp />
    </TotalAmountProvider>
  </SocketProvider>
);
```

```tsx
// SomeComponent.tsx
import React, { useEffect } from 'react';
import { useSocket } from '../context/socketContext';

const SomeComponent = () => {
  const socket = useSocket();

  useEffect(() => {
    if (socket) {
      socket.on('someEvent', (data) => {
        console.log(data);
      });
    }

    return () => {
      if (socket) {
        socket.off('someEvent');
      }
    };
  }, [socket]);

  return (
    <div>
      {/* Component content */}
    </div>
  );
};

export default SomeComponent;
```

8. Conclusion

Le fichier `socketContext.tsx` fournit une solution simple et efficace pour gérer la connexion WebSocket dans une application React Native. En encapsulant la logique de connexion WebSocket dans un contexte, il permet aux composants de l'application d'accéder facilement à cette connexion et de réagir aux événements WebSocket de manière centralisée et organisée.