UNIVERSIDADE ESTÁCIO DE SÁ

SISTEMAS DE INFORMAÇÃO

UNIDADE NOVA IGUAÇU

TRABALHO DE CRIAÇÃO APLICATIVO PLAYER DE MÚSICA EM ANDROID

NOVA IGUAÇU - RJ

2025

JOSÉ LUIS DE OLIVEIRA JUNIOR – 202308291068
GUILHERME MATEUS GENTA DA SILVA – 202308294407
MARIA BEATRIZ ALMEIDA CARDOSO – 202308296337
MATHEUS DE AGUIAR GERALDO – 202302610391
MIGUEL DOS SANTOS DE REZENDE – 202408496851

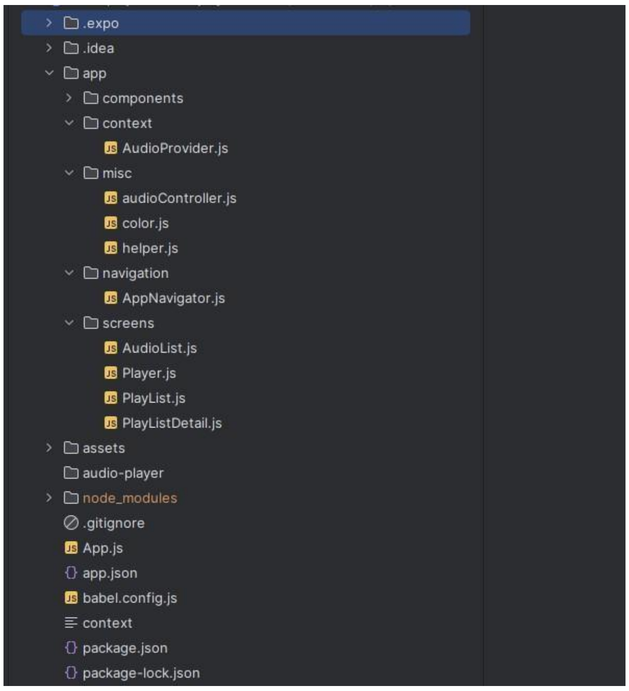
TRABALHO DE CRIAÇÃO **APLICATIVO PLAYER DE MÚSICA EM ANDROID**

PROJETO PARA CRIAÇÃO DE APLICATIVO PLAYER DE MÚSICA EM ANDROID COMO REQUESITO PARA OBTENÇÃO DE NOTA FINAL DO CURSO DE PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS EM ANDROID

ORIENTADOR: RONALDO SANTOS.

NOVA IGUAÇU – RJ
2025

Projeto EPlayer



```
> .expo
> .idea
v app
  > components
  v context
      AudioProvider.js
  v misc
      audioController.js
      color.js
      helper.js
  v navigation
      AppNavigator.js
  v screens
      AudioList.js
      Player.js
      PlayList.js
      PlayListDetail.js
> assets
  audio-player
> node_modules
  .gitignore
  App.js
  app.json
  babel.config.js
  context
  package.json
  package-lock.json
```

APP.JS

```
import React from "react";
import { NavigationContainer, DefaultTheme } from
"@react-navigation/native";
import AppNavigation from "./app/navigation/AppNavigator";
import AudioProvider from "./app/context/AudioProvider";
import color from "./app/misc/color";

// Tema customizado para o App
const MyTheme = {
 ...DefaultTheme,
 colors: {
   ...DefaultTheme.colors,
   background: color.APP_BG,
 },
};

// Componente principal do App
export default function App() {
 return (
   <AudioProvider>
     <NavigationContainer theme={MyTheme}>
       <AppNavigation />
     </NavigationContainer>
   </AudioProvider>
 );
}
```

PASTA APP (./app)

PASTA COMPONENTS (app/components)

AUDIOLISTITEM.JS

```
import React from "react";
import {
 View,
 StyleSheet,
 Text,
 Dimensions,
 TouchableWithoutFeedback,
} from "react-native";
import Entypo from "@expo/vector-icons/Entypo";
import color from "../misc/color";

const getThumbnailText = (filename) => filename[0];

const convertTime = (minutes) => {
 if (minutes) {
   const hrs = minutes / 60;
   const minute = hrs.toString().split(".")[0];
   const percent = parseInt(hrs.toString().split(".")[1].slice(0, 2));
   const sec = Math.ceil((60 * percent) / 100);

   if (parseInt(minute) < 10 && sec < 10) {
     return `0${minute}:0${sec}`;
   }

   if (parseInt(minute) < 10) {
     return `0${minute}:${sec}`;
   }

   if (sec < 10) {
     return `${minute}:0${sec}`;
   }

   return `${minute}:${sec}`;
 }
};

const renderPlayPauseIcon = (isPlaying) => {
 if (isPlaying)
   return (
     <Entypo name="controller-paus" size={24} color={color.ACTIVE_FONT} />
   );
 return <Entypo name="controller-play" size={24} color={color.ACTIVE_FONT}
/>;
};
```

```
const AudioListItem = ({
 title,
 duration,
 onOptionPress,
 onAudioPress,
 isPlaying,
 activeListItem,
}) => {
 return (
   <>
     <View style={styles.container}>
       <TouchableWithoutFeedback onPress={onAudioPress}>
         <View style={styles.leftContainer}>
           <View
             style={[
               styles.thumbnail,
               {
                 backgroundColor: activeListItem
                   ? color.ACTIVE_BG
                   : color.FONT_LIGHT,
               },
             ]}
           >
             <Text style={styles.thumbnailText}>
               {activeListItem
                 ? renderPlayPauseIcon(isPlaying)
                 : getThumbnailText(title)}
             </Text>
           </View>

           <View style={styles.titleContainer}>
             <Text numberOfLines={1} style={styles.title}>
               {title}
             </Text>
             <Text style={styles.timeText}>{convertTime(duration)}</Text>
           </View>
         </View>
       </TouchableWithoutFeedback>
       <View style={styles.rightContainer}>
         <Entypo
           onPress={onOptionPress}
           name="dots-three-vertical"
           size={20}
           color={color.FONT_MEDIUM}
           style={{ padding: 10 }}
         />
       </View>
     </View>
     <View style={styles.separator} />
   </>
 );
};
```

```
const { width } = Dimensions.get("window");
const styles = StyleSheet.create({
 container: {
   flexDirection: "row",
   alignSelf: "center",
   width: width - 80,
 },
 leftContainer: {
   flexDirection: "row",
   alignItems: "center",
   flex: 1,
 },
 rightContainer: {
   flexBasis: 50,
   height: 50,
   alignItems: "center",
   justifyContent: "center",
 },
 thumbnail: {
   height: 50,
   flexBasis: 50,
   backgroundColor: color.FONT_LIGHT,
   justifyContent: "center",
   alignItems: "center",
   borderRadius: 25,
 },
 thumbnailText: {
   fontSize: 22,
   fontWeight: "bold",
   color: color.FONT,
 },
 titleContainer: {
   width: width - 180,
   paddingLeft: 10,
 },
 title: {
   fontSize: 16,
   color: color.FONT,
 },
 separator: {
   width: width - 80,
   backgroundColor: "#333",
   opacity: 0.3,
   height: 0.5,
   alignSelf: "center",
   marginTop: 10,
 },
 timeText: {
   fontSize: 14,
   color: color.FONT_LIGHT,
 },
});
```

```
export default AudioListItem;
```

OPTIONMODAL.JS

```
import React from "react";
import {
 View,
 Text,
 StyleSheet,
 Modal,
 StatusBar,
 TouchableWithoutFeedback,
} from "react-native";
import color from "../misc/color";

const OptionModal = ({
 visible,
 currentItem,
 onClose,
 options,
 onPlayPress,
 onPlayListPress,
}) => {
 const { filename } = currentItem;
 return (
   <>
     <StatusBar hidden />
     <Modal animationType="slide" transparent={true} visible={visible}>
       <View style={styles.modal}>
         <Text style={styles.title} numberOfLines={2}>
           {filename}
         </Text>
         <View style={styles.optionContainer}>
           {options.map((optn) => {
             return (
               <TouchableWithoutFeedback
                 key={optn.title}
                 onPress={optn.onPress}
               >
                 <Text style={styles.option}>{optn.title}</Text>
               </TouchableWithoutFeedback>
             );
           })}
           {/* <TouchableWithoutFeedback onPress={onPlayPress}>
             <Text style={styles.option}>Play</Text>
           </TouchableWithoutFeedback>
           <TouchableWithoutFeedback onPress={onPlayListPress}>
             <Text style={styles.option}>Add à Playlist</Text>
           </TouchableWithoutFeedback> */}
         </View>
       </View>
       <TouchableWithoutFeedback onPress={onClose}>
```

```
         <View style={styles.modalBg} />
       </TouchableWithoutFeedback>
     </Modal>
   </>
 );
};

const styles = StyleSheet.create({
 modal: {
   position: "absolute",
   bottom: 0,
   right: 0,
   left: 0,
   backgroundColor: color.APP_BG,
   borderTopRightRadius: 20,
   borderTopLeftRadius: 20,
   zIndex: 1000,
 },
 optionContainer: {
   padding: 20,
 },
 title: {
   fontSize: 18,
   fontWeight: "bold",
   padding: 20,
   paddingBottom: 0,
   color: color.FONT_MEDIUM,
 },
 option: {
   fontSize: 16,
   fontWeight: "bold",
   color: color.FONT,
   paddingVertical: 10,
   letterSpacing: 1,
 },
 modalBg: {
   position: "absolute",
   top: 0,
   right: 0,
   left: 0,
   bottom: 0,
   backgroundColor: color.MODAL_BG,
 },
});

export default OptionModal;
```

PLAYBUTTON.JS

```
import React from "react";
import { AntDesign } from "@expo/vector-icons";
import color from "../misc/color";
```

```
const PlayerButton = (props) => {
 const { iconType, size = 50, iconColor = color.FONT, onPress } = props;

 const getIconName = (type) => {
   switch (type) {
     case "PLAY":
       return "pausecircle";
     case "PAUSE":
       return "playcircleo";
     case "NEXT":
       return "forward";
     case "PREV":
       return "banckward";
   }
 };
 return (
   <AntDesign
     {...props}
     onPress={onPress}
     name={getIconName(iconType)}
     size={size}
     color={iconColor}
   />
 );
};

export default PlayerButton;
```

PLAYLISTDETAIL.JS

```
import React from "react";
import {
 View,
 StyleSheet,
 Modal,
 FlatList,
 Text,
 Dimensions,
} from "react-native";
import color from "../misc/color";
import AudioListItem from "./AudioListItem";
import { selectAudio } from "../misc/audioController";

const PlayListDetail = ({ visible, playList, onClose }) => {
 const playAudio = (audio) => {
   selectAudio(audio);
 };

  return (
   <Modal
     visible={visible}
     animationType="slide"
     transparent
     onRequestClose={onClose}
   >
     <View style={styles.container}>
       <Text style={styles.title}>{playList.title}</Text>
       <FlatList
         contentContainerStyle={styles.listContainer}
         data={playList.audios}
         keyExtractor={(item) => item.id.toString()}
         renderItem={({ item }) => (
           <View style={{ marginBottom: 10 }}>
             <AudioListItem
               title={item.filename}
               duration={item.duration}
               onAudioPress={() => playAudio(item)}
             />
           </View>
         )}
       />
     </View>
     <View style={[StyleSheet.absoluteFillObject, styles.modalBG]} />
   </Modal>
 );
};

const { width, height } = Dimensions.get("window");

const styles = StyleSheet.create({
 container: {
   position: "absolute",
   bottom: 0,
   alignSelf: "center",
   height: height - 230,
   width: width - 15,
   backgroundColor: color.ACTIVE_FONT,
   borderTopRightRadius: 30,
   borderTopLeftRadius: 30,
 },
 modalBG: {
   backgroundColor: color.MODAL_BG,
   zIndex: -1,
 },
 title: {
   textAlign: "center",
   fontSize: 20,
   paddingVertical: 5,
   fontWeight: "bold",
   color: color.ACTIVE_BG,
 },
 listContainer: {
   padding: 20,
 },
});

export default PlayListDetail;
```

PLAYLISTINPUTMODAL.JS

```
import { React, useState } from "react";
import {
 View,
 StyleSheet,
 Modal,
 TextInput,
 Dimensions,
 TouchableWithoutFeedback,
 Text,
} from "react-native";
import { AntDesign } from "@expo/vector-icons";
import color from "../misc/color";

const PlayListInputModal = ({ visible, onClose, onSubmit }) => {
 const [playListName, setPlayListName] = useState("");

 const handleOnSubmit = () => {
   if (!playListName.trim()) {
     onClose();
   } else {
     onSubmit(playListName);
     setPlayListName("");
     onClose();
   }
 };

 return (
   <Modal visible={visible} animationType="fade" transparent>
     <View style={styles.modalContainer}>
       <View style={styles.inputContainer}>
         <Text style={{ color: color.ACTIVE_BG, fontSize: 16 }}>
           Criar nova PlayList
         </Text>
         <TextInput
           value={playListName}
           onChangeText={(text) => setPlayListName(text)}
           style={styles.input}
         />
         <AntDesign
           name="check"
           size={24}
           color={color.ACTIVE_FONT}
           style={styles.submitIcon}
           onPress={handleOnSubmit}
         />
       </View>
     </View>
     <TouchableWithoutFeedback onPress={onClose}>
       <View style={[StyleSheet.absoluteFillObject, styles.modalBG]} />
     </TouchableWithoutFeedback>
   </Modal>
 );
};

const { width } = Dimensions.get("window");
const styles = StyleSheet.create({
 modalContainer: {
   flex: 1,
   justifyContent: "center",
   alignItems: "center",
 },
 inputContainer: {
   width: width - 20,
   height: 200,
   borderRadius: 10,
   backgroundColor: color.ACTIVE_FONT,
   justifyContent: "center",
   alignItems: "center",
 },
 input: {
   width: width - 40,
   borderBottomWidth: 1,
   borderBottomColor: color.ACTIVE_BG,
   fontSize: 18,
   paddingVertical: 5,
 },
 submitIcon: {
   padding: 10,
   backgroundColor: color.ACTIVE_BG,
   borderRadius: 50,
   marginTop: 15,
 },
 modalBG: {
   backgroundColor: color.MODAL_BG,
   zIndex: -1,
 },
});

export default PlayListInputModal;
```

SCREEN.JS

```
import React from "react";
import { View, StyleSheet, StatusBar } from "react-native";
import color from "../misc/color";

const Screen = ({ children }) => {
 return <View style={styles.container}>{children}</View>;
};

const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: color.APP_BG,
   paddingTop: StatusBar.currentHeight,
 },
});

export default Screen;
```

PASTA CONTEXT (app/context)

AUDIOPROVIDER.JS

```
import React, { Component, createContext } from "react";
import { View, Text, Alert } from "react-native";
import * as MediaLibrary from "expo-media-library";
import { DataProvider } from "recyclerlistview";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { Audio } from "expo-av";
import { storeAudioForNextOpening } from "../misc/helper";
import { playNext } from "../misc/audioController";

// Criação do contexto de áudio
export const AudioContext = createContext();

export class AudioProvider extends Component {
 constructor(props) {
   super(props);
   this.state = {
     audioFiles: [],
     playList: [],
     addToPlayList: null,
     permissionError: false,
     dataProvider: new DataProvider((r1, r2) => r1 !== r2),
     playbackObj: null,
     soundObj: null,
     currentAudio: {},
     isPlaying: false,
     isPlayListRunning: false,
     activePlayList: [],
     currentAudioIndex: null,
     playbackPosition: null,
     playbackDuration: null,
   };
   this.totalAudioCount = 0;
 }

 // Alerta de permissão para acessar os arquivos do dispositivo
 permissionAlert = () => {
   Alert.alert(
     "Permissão Requerida",
     "Essa aplicação precisa de acesso aos aquivos de áudio!",
     [
       { text: "Conceder", onPress: () => this.getPermission() },
       { text: "Cancelar", onPress: () => this.permissionAlert() },
     ]
   );
 };

 // Função para obter arquivos de áudio
 getAudioFiles = async () => {
   const { dataProvider, audioFiles } = this.state;
   let media = await MediaLibrary.getAssetsAsync({
     mediaType: "audio",
   });
   media = await MediaLibrary.getAssetsAsync({
     mediaType: "audio",
     first: media.totalCount,
   });
   this.totalAudioCount = media.totalCount;

   this.setState({
     ...this.state,
     dataProvider: dataProvider.cloneWithRows([
       ...audioFiles,
       ...media.assets,
     ]),
     audioFiles: [...audioFiles, ...media.assets],
   });
 };

 // Função para carregar o áudio anterior
 loadPreviousAudio = async () => {
   let previousAudio = await AsyncStorage.getItem("previousAudio");
   let currentAudio;
   let currentAudioIndex;

   if (previousAudio === null) {
     currentAudio = this.state.audioFiles[0];
     currentAudioIndex = 0;
   } else {
     previousAudio = JSON.parse(previousAudio);
     currentAudio = previousAudio.audio;
     currentAudioIndex = previousAudio.index;
   }
   this.setState({ ...this.state, currentAudio, currentAudioIndex });
 };

 // Função para obter permissão
 getPermission = async () => {
   const permission = await MediaLibrary.getPermissionsAsync();
   if (permission.granted) {
     // Transmitir para o app todos os arquivos de áudio
     this.getAudioFiles();
   }

   if (!permission.canAskAgain && !permission.granted) {
     this.setState({ ...this.state, permissionError: true });
   }

   if (!permission.granted && permission.canAskAgain) {
     const { status, canAskAgain } =
       await MediaLibrary.requestPermissionsAsync();
     if (status === "denied" && canAskAgain) {
       // Exibir alerta dizendo que o usuário precisa conceder permissão
para que o app funcione como planejado
       this.permissionAlert();
     }

     if (status === "granted") {
       // Transmitir para o app todos os arquivos de áudio
       this.getAudioFiles();
     }

     if (status === "denied" && !canAskAgain) {
       //  será transmitido um erro ao usuário
       this.setState({ ...this.state, permissionError: true });
     }
   }
 };

 // Função para atualizar o status da reprodução
 onPlaybackStatusUpdate = async (playbackStatus) => {
   if (playbackStatus.isLoaded && playbackStatus.isPlaying) {
     this.updateState(this, {
       playbackPosition: playbackStatus.positionMillis,
       playbackDuration: playbackStatus.durationMillis,
     });
   }

   if (playbackStatus.isLoaded && !playbackStatus.isPlaying) {
     storeAudioForNextOpening(
       this.state.currentAudio,
       this.state.currentAudioIndex,
       playbackStatus.positionMillis
     );
   }

   if (playbackStatus.didJustFinish) {
     if (this.state.isPlayListRunning) {
       let audio;
       const indexOnPlayList = this.state.activePlayList.audios.findIndex(
         ({ id }) => id === this.state.currentAudio.id
       );
       const nextIndex = indexOnPlayList + 1;
```

```
      audio = this.state.activePlayList.audios[nextIndex];

      if (!audio) audio = this.state.activePlayList.audios[0];

      const indexOnAllList = this.state.audioFiles.findIndex(
        ({ id }) => id === audio.id
      );

      const status = await playNext(this.state.playbackObj, audio.uri);
      return this.updateState(this, {
        soundObj: status,
        isPlaying: true,
        currentAudio: audio,
        currentAudioIndex: indexOnAllList,
      });
    }

    const nextAudioIndex = this.state.currentAudioIndex + 1;

    //Caso não haja próximo áudio para tocar ou o áudio atual é o último
    if (nextAudioIndex >= this.totalAudioCount) {
      this.state.playbackObj.unloadAsync();
      this.updateState(this, {
        soundObj: null,
        currentAudio: this.state.audioFiles[0],
        isPlaying: false,
        currentAudioIndex: 0,
        playbackPosition: null,
        playbackDuration: null,
      });
      return await storeAudioForNextOpening(this.state.audioFiles[0], 0);
    }

    //Caso contrário será selecionado o próximo áudio
    const audio = this.state.audioFiles[nextAudioIndex];
    const status = await playNext(this.state.playbackObj, audio.uri);
    this.updateState(this, {
      soundObj: status,
      currentAudio: audio,
      isPlaying: true,
      currentAudioIndex: nextAudioIndex,
    });
    await storeAudioForNextOpening(audio, nextAudioIndex);
  }
};

componentDidMount() {
  this.getPermission();
  if (this.state.playbackObj === null) {
    this.setState({ ...this.state, playbackObj: new Audio.Sound() });
  }
}
```

```
// Função para atualizar
updateState = (prevState, newState = {}) => {
  this.setState({ ...prevState, ...newState });
};

render() {
  const {
    audioFiles,
    playList,
    addToPlayList,
    dataProvider,
    permissionError,
    playbackObj,
    soundObj,
    currentAudio,
    isPlaying,
    currentAudioIndex,
    playbackPosition,
    playbackDuration,
    isPlayListRunning,
    activePlayList,
  } = this.state;
  if (permissionError)
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center",
        }}
      >
        <Text style={{ fontSize: 25, textAlign: "center", color: "red" }}>
          Parece que você não aceitou a permissão.
        </Text>
      </View>
    );
  return (
    <AudioContext.Provider
      value={{
        audioFiles,
        playList,
        addToPlayList,
        dataProvider,
        playbackObj,
        soundObj,
        currentAudio,
        isPlaying,
        currentAudioIndex,
        totalAudioCount: this.totalAudioCount,
        playbackPosition,
        playbackDuration,
        isPlayListRunning,
        activePlayList,
         updateState: this.updateState,
         loadPreviousAudio: this.loadPreviousAudio,
         onPlaybackStatusUpdate: this.onPlaybackStatusUpdate,
       }}
     >
       {this.props.children}
     </AudioContext.Provider>
   );
 }
}

export default AudioProvider;
```

PASTA MISC (app/misc)

AUDIOCONTROLLER.JS

```
import { storeAudioForNextOpening } from "./helper";

// Função para tocar música
export const play = async (playbackObj, uri, lastPosition) => {
 try {
   //Se não tiver última posição
   if (!lastPosition)
     return await playbackObj.loadAsync(
       { uri },
       { shouldPlay: true, progressUpdateIntervalMillis: 1000 }
     );

   //Mas se tiver última posição, então será tocado o áudio da última
posição
   await playbackObj.loadAsync(
     { uri },
     { progressUpdateIntervalMillis: 1000 }
   );

   return await playbackObj.playFromPositionAsync(lastPosition);
 } catch (error) {
   console.log("erro dento do metodo de ajuda play", error.message);
 }
};

// Função para pausar música
export const pause = async (playbackObj) => {
 try {
   return await playbackObj.setStatusAsync({ shouldPlay: false });
 } catch (error) {
   console.log("erro dentro do metodo de ajuda pause", error.message);
 }
};

// Função para retomar música
```

```
export const resume = async (playbackObj) => {
 try {
   return await playbackObj.playAsync();
 } catch (error) {
   console.log("erro dentro do metodo de ajuda resume", error.message);
  }
};

// Função para selecionar outra música
export const playNext = async (playbackObj, uri) => {
 try {
   await playbackObj.stopAsync();
   await playbackObj.unloadAsync();
   return await play(playbackObj, uri);
 } catch (error) {
   console.log("erro dentro do metodo de ajuda playNext", error.message);
  }
};

// Função para selecionar áudio
export const selectAudio = async (audio, context, playListInfo = {}) => {
 const {
   soundObj,
   playbackObj,
   currentAudio,
   updateState,
   audioFiles,
   onPlaybackStatusUpdate,
 } = context;

 try {
   //Tocar audio pela primeira vez
   if (soundObj === null) {
     const status = await play(playbackObj, audio.uri, audio.lastPosition);
     const index = audioFiles.findIndex(({ id }) => id === audio.id);
     updateState(context, {
       currentAudio: audio,
       soundObj: status,
       isPlaying: true,
       currentAudioIndex: index,
       isPlayListRunning: false,
       activePlayList: [],
       ...playListInfo,
     });
     playbackObj.setOnPlaybackStatusUpdate(onPlaybackStatusUpdate);
     return storeAudioForNextOpening(audio, index);
   }

   //Pausar o audio
   if (
     soundObj.isLoaded &&
     soundObj.isPlaying &&
     currentAudio.id === audio.id
   ) {
     const status = await pause(playbackObj);
     return updateState(context, {
       soundObj: status,
       isPlaying: false,
       playbackPosition: status.positionMillis,
     });
   }

   //Retomar o audio
   if (
     soundObj.isLoaded &&
     !soundObj.isPlaying &&
     currentAudio.id === audio.id
   ) {
     const status = await resume(playbackObj);
     return updateState(context, { soundObj: status, isPlaying: true });
   }

   // Tocar outro áudio
   if (soundObj.isLoaded && currentAudio.id !== audio.id) {
     const status = await playNext(playbackObj, audio.uri);
     const index = audioFiles.findIndex(({ id }) => id === audio.id);
     updateState(context, {
       currentAudio: audio,
       soundObj: status,
       isPlaying: true,
       currentAudioIndex: index,
       isPlayListRunning: false,
       activePlayList: [],
       ...playListInfo,
     });
     return storeAudioForNextOpening(audio, index);
   }
 } catch (error) {
   console.log("Erro dentro do método de seleção de áudio.",
error.message);
 }
};

// Função para selecionar áudio da playlist
const selectAudioFromPlayList = async (context, select) => {
 const { activePlayList, currentAudio, audioFiles, playbackObj, updateState
} =
   context;

 let audio;
 let defaultIndex;
 let nextIndex;

 const indexOnPlayList = activePlayList.audios.findIndex(
   ({ id }) => id === currentAudio.id
 );
 if (select === "next") {
   nextIndex = indexOnPlayList + 1;
   defaultIndex = 0;
 }

 if (select === "previous") {
   nextIndex = indexOnPlayList - 1;
   defaultIndex = activePlayList.audios.length - 1;
 }

 audio = activePlayList.audios[nextIndex];

 if (!audio) audio = activePlayList.audios[defaultIndex];

 const indexOnAllList = audioFiles.findIndex(({ id }) => id === audio.id);

 const status = await playNext(playbackObj, audio.uri);
 return updateState(context, {
   soundObj: status,
   isPlaying: true,
   currentAudio: audio,
   currentAudioIndex: indexOnAllList,
 });
};

// Função para mudar o áudio
export const changeAudio = async (context, select) => {
 const {
   playbackObj,
   currentAudioIndex,
   totalAudioCount,
   audioFiles,
   updateState,
   isPlayListRunning,
 } = context;

 if (isPlayListRunning) return selectAudioFromPlayList(context, select);

 try {
   const { isLoaded } = await playbackObj.getStatusAsync();
   const isLastAudio = currentAudioIndex + 1 === totalAudioCount;
   const isFirstAudio = currentAudioIndex <= 0;
   let  audio;
   let  index;
   let status;

   // Próximo
   if (select === "next") {
     audio = audioFiles[currentAudioIndex + 1];
     if (!isLoaded && !isLastAudio) {
       index = currentAudioIndex + 1;
       status = await play(playbackObj, audio.uri);
```

```
      playbackObj.setOnPlaybackStatusUpdate(onPlaybackStatusUpdate);
    }

    if (isLoaded && !isLastAudio) {
      index = currentAudioIndex + 1;
      status = await playNext(playbackObj, audio.uri);
    }

    if (isLastAudio) {
      index = 0;
      audio = audioFiles[index];
      if (isLoaded) {
        status = await playNext(playbackObj, audio.uri);
      } else {
        status = await play(playbackObj, audio.uri);
      }
    }
  }

  // Anterior
  if (select === "previous") {
    audio = audioFiles[currentAudioIndex - 1];
    if (!isLoaded && !isFirstAudio) {
      index = currentAudioIndex - 1;
      status = await play(playbackObj, audio.uri);
      playbackObj.setOnPlaybackStatusUpdate(onPlaybackStatusUpdate);
    }

    if (isLoaded && !isFirstAudio) {
      index = currentAudioIndex - 1;
      status = await playNext(playbackObj, audio.uri);
    }

    if (isFirstAudio) {
      index = totalAudioCount - 1;
      audio = audioFiles[index];
      if (isLoaded) {
        status = await playNext(playbackObj, audio.uri);
      } else {
        status = await play(playbackObj, audio.uri);
      }
    }
  }

  updateState(context, {
    currentAudio: audio,
    soundObj: status,
    isPlaying: true,
    currentAudioIndex: index,
    playbackPosition: null,
    playbackDuration: null,
  });
  storeAudioForNextOpening(audio, index);
```

```
 } catch (error) {
   console.log("Erro dentro do método de troca de áudio.", error.message);
 }
};

// Função para mover o áudio
export const moveAudio = async (context, value) => {
 const { soundObj, isPlaying, playbackObj, updateState } = context;
 if (soundObj === null || !isPlaying) return;

 try {
   const status = await playbackObj.setPositionAsync(
     Math.floor(soundObj.durationMillis * value)
   );
   updateState(context, {
     soundObj: status,
     playbackPosition: status.positionMillis,
   });

   await resume(playbackObj);
 } catch (error) {
   console.log("Erro dentro da chamada onSlidingComplete", error);
 }
};
```

COLOR.JS

```
export default {
 APP_BG: "#191414", // Spotify background color
 FONT: "#FFFFFF", // Spotify font color
 FONT_MEDIUM: "#FFFFFF", // Spotify medium font color
 FONT_LIGHT: "#1DB954", // Spotify light font color
 MODAL_BG: "rgba(0, 0, 0, 0.8)", // Spotify modal background color
 ACTIVE_BG: "#1DB954", // Spotify active background color
 ACTIVE_FONT: "#FFFFFF", // Spotify active font color
 MODAL_LIST: "rgba(40, 40, 40, 0.8)", // Spotify modal list color
};
```

HELPER.JS

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

// Função para armazenar o áudio para a próxima abertura do aplicativo
export const storeAudioForNextOpening = async (audio, index, lastPosition)
=> {
 await AsyncStorage.setItem(
   "previousAudio",
   JSON.stringify({ audio: { ...audio, lastPosition }, index })
 );
};
```

```
// Função para converter o tempo de minutos para o formato mm:ss
export const convertTime = (minutes) => {
 if (minutes) {
   const hrs = minutes / 60; // Converte minutos para horas
   const minute = hrs.toString().split(".")[0]; // Obtém a parte inteira
das horas
   const percent = parseInt(hrs.toString().split(".")[1].slice(0, 2)); //
Obtém os primeiros dois dígitos da parte decimal
   const sec = Math.ceil((60 * percent) / 100); // Converte a parte decimal
para segundos

   // Formata o tempo para mm:ss
   if (parseInt(minute) < 10 && sec < 10) {
     return `0${minute}:0${sec}`;
   }

   if (sec == 60) {
     return `${minute + 1}:00`;
   }

   if (parseInt(minute) < 10) {
     return `0${minute}:${sec}`;
   }

   if (sec < 10) {
     return `${minute}:0${sec}`;
   }

   return `${minute}:${sec}`;
 }
};
```

PASTA NAVIGATION (app/navigation)

APPNAVIGATOR.JS

```
import React from "react";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { createStackNavigator } from "@react-navigation/stack";
import AudioList from "../screens/AudioList";
import Player from "../screens/Player";
import PlayList from "../screens/PlayList";
import Ionicons from "@expo/vector-icons/Ionicons";
import FontAwesome5 from "@expo/vector-icons/FontAwesome5";
import MaterialIcons from "@expo/vector-icons/MaterialIcons";
import PlayListDetail from "../screens/PlayListDetail";

const Tab = createBottomTabNavigator();
const Stack = createStackNavigator();

// Stack navigator para telas da PlayList
```

```
const PlayListScreen = () => {
 return (
   <Stack.Navigator screenOptions={{ headerShown: false }}>
     <Stack.Screen name="PlayList" component={PlayList} />
     <Stack.Screen name="PlayListDetail" component={PlayListDetail} />
   </Stack.Navigator>
 );
};

// Navegação principal do aplicativo
const AppNavigation = () => {
 return (
   <Tab.Navigator
     screenOptions={({ route }) => ({
       tabBarIcon: ({ color, size }) => {
         let iconName;

         // Definindo ícones para cada aba
         if (route.name === "AudioList") {
           iconName = "headset";
           return <Ionicons name={iconName} size={size} color={color} />;
         } else if (route.name === "Player") {
           iconName = "compact-disc";
           return <FontAwesome5 name={iconName} size={size} color={color}
/>;
         } else if (route.name === "PlayListScreen") {
           iconName = "library-music";
           return <MaterialIcons name={iconName} size={size} color={color}
/>;
         }
       },
       //Definindo cores para a barra onde ficam os ícones de tela
       tabBarActiveTintColor: "green",
       tabBarInactiveTintColor: "white",
       tabBarStyle: {
         backgroundColor: "black",
       },
     })}
   >
     <Tab.Screen
       name="AudioList"
       component={AudioList}
       options={{ headerShown: false }}
     />
     <Tab.Screen
       name="Player"
       component={Player}
       options={{ headerShown: false }}
     />
     <Tab.Screen
       name="PlayListScreen"
       component={PlayListScreen}
       options={{ headerShown: false }}
     />
   </Tab.Navigator>
 );
};

export default AppNavigation;
```

PASTA SCREENS (app/screens)

AUDIOLIST.JS

```
import React, { Component } from "react";
import { Text, View, StyleSheet, Dimensions, TextInput } from
"react-native";
import { AudioContext } from "../context/AudioProvider";
import {
 RecyclerListView,
 LayoutProvider,
 DataProvider,
} from "recyclerlistview";
import AudioListItem from "../components/AudioListItem";
import Screen from "../components/Screen";
import OptionModal from "../components/OptionModal";
import { Audio } from "expo-av";
import { Ionicons } from "@expo/vector-icons";
import {
 play,
 pause,
 resume,
 playNext,
 selectAudio,
} from "../misc/audioController";
import { storeAudioForNextOpening } from "../misc/helper";

export class AudioList extends Component {
 static contextType = AudioContext;

 constructor(props) {
   super(props);
   this.state = {
     optionModalVisible: false,
     searchQuery: "",
     filteredAudioFiles: new DataProvider((r1, r2) => r1 !== r2),
   };
   this.currentItem = {};
 }

 // LayoutProvider para definir o layout dos itens da lista
 layoutProvider = new LayoutProvider(
   (i) => "audio",
   (type, dim) => {
     switch (type) {
       case "audio":
         dim.width = Dimensions.get("window").width;
         dim.height = 70;
         break;
       default:
         dim.width = 0;
         dim.height = 0;
     }
   }
 );

 // Função para lidar com o clique no áudio
 handleAudioPress = async (audio) => {
   await selectAudio(audio, this.context);
 };

 // Carregar o áudio anterior
 componentDidMount() {
   this.context.loadPreviousAudio();
 }

 componentDidMount() {
   this.context.loadPreviousAudio();
   this.setState({
     filteredAudioFiles: this.state.filteredAudioFiles.cloneWithRows(
       this.context.audioFiles
     ),
   });
 }

 // Função para lidar com a pesquisa
 handleSearch = (text) => {
   const filteredAudioFiles = this.context.audioFiles.filter((audio) =>
     audio.filename.toLowerCase().includes(text.toLowerCase())
   );
   this.setState({
     searchQuery: text,
     filteredAudioFiles:
       this.state.filteredAudioFiles.cloneWithRows(filteredAudioFiles),
   });
 };

 // Função para renderizar cada item da lista
 rowRenderer = (type, item, index, extendedState) => {
   return (
     <AudioListItem
       title={item.filename}
       isPlaying={extendedState.isPlaying}
       activeListItem={this.context.currentAudioIndex === index}
       duration={item.duration}
       onAudioPress={() => this.handleAudioPress(item)}
       onOptionPress={() => {
         this.currentItem = item;
```

```
          this.setState({ ...this.state, optionModalVisible: true });
        }}
      />
    );
  };

  // Navegar para a tela de PlayList
  navigateToPlaylist = () => {
    this.context.updateState(this.context, {
      addToPlayList: this.currentItem,
    });
    this.props.navigation.navigate("PlayList");
  };

  render() {
    return (
      <AudioContext.Consumer>
        {({ isPlaying, audioFiles }) => {
          if (!audioFiles.length) return null;
          return (
            <Screen>
              <View style={styles.searchContainer}>
                <Ionicons name="search" size={24} color="black" />
                <TextInput
                  style={styles.searchInput}
                  placeholder="Pesquisar..."
                  value={this.state.searchQuery}
                  onChangeText={this.handleSearch}
                />
              </View>
              <RecyclerListView
                dataProvider={this.state.filteredAudioFiles}
                layoutProvider={this.layoutProvider}
                rowRenderer={this.rowRenderer}
                extendedState={{ isPlaying }}
              />
              <OptionModal
                options={[
                  { title: "Add à PlayList", onPress:
this.navigateToPlaylist },
                ]}
                currentItem={this.currentItem}
                onClose={() =>
                  this.setState({ ...this.state, optionModalVisible: false
})
                }
                visible={this.state.optionModalVisible}
              />
            </Screen>
          );
        }}
      </AudioContext.Consumer>
    );
```

```
 }
}

// Estilos para o componente
const styles = StyleSheet.create({
 container: {
   flex: 1,
   justifyContent: "center",
   alignItems: "center",
 },
 searchContainer: {
   flexDirection: "row",
   alignItems: "center",
   padding: 10,
   backgroundColor: "#fff",
   borderRadius: 10,
   margin: 10,
 },
 searchInput: {
   flex: 1,
   marginLeft: 10,
   fontSize: 16,
 },
});

export default AudioList;
```

PLAYER.JS

```
import React, { useContext, useEffect, useState, useRef } from "react";
import {
 View,
 StyleSheet,
 Text,
 Dimensions,
 Animated,
 Easing,
} from "react-native";
import Screen from "../components/Screen";
import color from "../misc/color";
import MaterialCommunityIcons from
"@expo/vector-icons/MaterialCommunityIcons";
import Slider from "@react-native-community/slider";
import PlayerButton from "../components/PlayerButton";
import { AudioContext } from "../context/AudioProvider";
import {
 play,
 pause,
 resume,
 playNext,
 selectAudio,
 changeAudio,
 moveAudio,
} from "../misc/audioController";
import { convertTime, storeAudioForNextOpening } from "../misc/helper";

// Obtendo a largura da janela do dispositivo
const { width } = Dimensions.get("window");

const Player = () => {
 // Estado para armazenar a posição atual do áudio
 const [currentPosition, setCurrentPosition] = useState(0);
 const context = useContext(AudioContext);
 const { playbackPosition, playbackDuration, currentAudio } = context;

 // Valor de rotação para animação
 const rotateValue = useRef(new Animated.Value(0)).current;

 // Função para calcular a posição da barra de progresso
 const calculateSeebBar = () => {
   if (playbackPosition !== null && playbackDuration !== null) {
     return playbackPosition / playbackDuration;
   }

   if (currentAudio.lastPosition) {
     return currentAudio.lastPosition / (currentAudio.duration * 1000);
   }
   return 0;
 };

 // Carregar o áudio anterior
 useEffect(() => {
   context.loadPreviousAudio();
 }, []);

 // Iniciar ou parar a animação de rotação com base no estado de reprodução
 useEffect(() => {
   if (context.isPlaying) {
     startImageRotateFunction();
   } else {
     rotateValue.stopAnimation();
   }
 }, [context.isPlaying]);

 // Função para iniciar a animação de rotação
 const startImageRotateFunction = () => {
   rotateValue.setValue(0);
   Animated.loop(
     Animated.timing(rotateValue, {
       toValue: 1,
       duration: 5000,
       easing: Easing.linear,
       useNativeDriver: true,
     })
   ).start();
```

```
};

// Função para lidar com o botão de play/pause
const handlePlayPause = async () => {
  await selectAudio(context.currentAudio, context);
};

// Função para avançar para a próxima música
const handleNext = async () => {
  await changeAudio(context, "next");
};

// Função para voltar para a música anterior
const handlePrevious = async () => {
  await changeAudio(context, "previous");
};

// Função para renderizar o tempo atual do áudio
const renderCurrentTime = () => {
  if (!context.soundObj && currentAudio.lastPosition) {
    return convertTime(currentAudio.lastPosition / 1000);
  }
  return convertTime(context.playbackPosition / 1000);
};

if (!context.currentAudio) return null;
return (
  <Screen name="Player">
    <View style={styles.container}>
      <View style={styles.audioCountContainer}>
        <View style={{ flexDirection: "row" }}>
          {context.isPlayListRunning && (
            <>
              <Text style={{ fontWeight: "bold", color: color.FONT_MEDIUM
}}>
                Música da PlayList:{" "}
              </Text>
              <Text style={{ color: color.FONT_MEDIUM }}>
                {context.activePlayList.title}
              </Text>
            </>
          )}
        </View>

        <Text style={styles.audioCount}>{`${
          context.currentAudioIndex + 1
        } / ${context.totalAudioCount}`}</Text>
      </View>

      <View style={styles.midBannerContainer}>
        <Animated.View
          style={{
            transform: [
              {
                rotate: rotateValue.interpolate({
                  inputRange: [0, 1],
                  outputRange: ["0deg", "360deg"],
                }),
              },
            ],
          }}
        >
          <MaterialCommunityIcons
            name="music-circle"
            size={350}
            color={context.isPlaying ? color.ACTIVE_BG :
color.FONT_MEDIUM}
          />
        </Animated.View>
      </View>
      <View style={styles.audioPlayerContainer}>
        <Text numberOfLines={1} style={styles.audioTitle}>
          {context.currentAudio.filename}
        </Text>

        <View
          style={{
            flexDirection: "row",
            justifyContent: "space-between",
            paddingHorizontal: 15,
            color: color.FONT_MEDIUM,
          }}
        >
          <Text style={{ color: color.FONT_MEDIUM }}>
            {convertTime(context.currentAudio.duration)}
          </Text>
          <Text style={{ color: color.FONT_MEDIUM }}>
            {currentPosition ? currentPosition : renderCurrentTime()}
          </Text>
        </View>

        <Slider
          style={{ width: width, height: 40 }}
          minimumValue={0}
          maximumValue={1}
          value={calculateSeebBar()}
          minimumTrackTintColor={color.FONT_MEDIUM}
          maximumTrackTintColor={color.ACTIVE_BG}
          onValueChange={(value) => {
            setCurrentPosition(
              convertTime(value * context.currentAudio.duration)
            );
          }}
          onSlidingStart={async () => {
            if (!context.isPlaying) return;
```

```
             try {
               await pause(context.playbackObj);
             } catch (error) {
               console.log("Erro dentro da chamada onSlidingStart", error);
             }
           }}
           onSlidingComplete={async (value) => {
             await moveAudio(context, value);
             setCurrentPosition(0);
           }}
         />

         <View style={styles.audioControllers}>
           <PlayerButton iconType="PREV" onPress={handlePrevious} />
           <PlayerButton
             onPress={handlePlayPause}
             style={{ marginHorizontal: 25 }}
             iconType={context.isPlaying ? "PLAY" : "PAUSE"}
           />
           <PlayerButton iconType="NEXT" onPress={handleNext} />
         </View>
       </View>
     </View>
   </Screen>
 );
};

// Estilos para o componente
const styles = StyleSheet.create({
 audioControllers: {
   width,
   flexDirection: "row",
   justifyContent: "center",
   alignItems: "center",
   paddingBottom: 20,
 },
 audioCountContainer: {
   flexDirection: "row",
   justifyContent: "space-between",
   paddingHorizontal: 15,
 },
 container: {
   flex: 1,
 },
 audioCount: {
   textAlign: "right",
   color: color.FONT_LIGHT,
   fontSize: 14,
 },
 midBannerContainer: {
   flex: 1,
   justifyContent: "center",
   alignItems: "center",
 },
 audioTitle: {
   fontSize: 20,
   color: color.ACTIVE_BG,
   padding: 15,
 },
});

export default Player;
```

PLAYLIST.JS

```
import React, { useContext, useEffect, useState } from "react";
import {
 View,
 StyleSheet,
 Text,
 ScrollView,
 TouchableOpacity,
 FlatList,
 Alert,
} from "react-native";
import color from "../misc/color";
import PlayListInputModal from "../components/PlayListInputModal";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { AudioContext } from "../context/AudioProvider";
import PlayListDetail from "../components/PlayListDetail";

// Variável para armazenar a playlist selecionada
let selectedPlayList = {};
const PlayList = ({ navigation }) => {
 // Controle de visibilidade do modal e da lista de reprodução
 const [modalVisible, setModalVisible] = useState(false);
 const [showPlayList, setShowPlayList] = useState(false);

 const context = useContext(AudioContext);
 const { playList, addToPlayList, updateState } = context;

 // Função para criar uma nova playlist
 const createPlayList = async (playListName) => {
   const result = await AsyncStorage.getItem("playlist");
   if (result !== null) {
     const audios = [];
     if (addToPlayList) {
       audios.push(addToPlayList);
     }
     const newList = {
       id: Date.now(),
       title: playListName,
       audios: audios,
     };
```

```
    const updatedList = [...playList, newList];
    updateState(context, { addToPlayList: null, playList: updatedList });
    await AsyncStorage.setItem("playlist", JSON.stringify(updatedList));
  }
  setModalVisible(false);
};

// Função para renderizar a lista de reprodução
const renderPlayList = async () => {
  const result = await AsyncStorage.getItem("playlist");
  if (result === null) {
    const defaultPlayList = {
      id: Date.now(),
      title: "Meu Favorito",
      audios: [],
    };

    const newPlayList = [...playList, defaultPlayList];
    updateState(context, {
      playList: [...newPlayList],
    });
    return await AsyncStorage.setItem(
      "playlist",
      JSON.stringify([...newPlayList])
    );
  }
  updateState(context, {
    playList: JSON.parse(result),
  });
};

// useEffect para carregar a lista de reprodução ao montar o componente
useEffect(() => {
  if (!playList.length) {
    renderPlayList();
  }
}, []);

// Função para lidar com o clique no banner da playlist
const handleBannerPress = async (playList) => {
  if (addToPlayList) {
    const result = await AsyncStorage.getItem("playlist");

    let  oldList  =  [];
    let updatedList = [];
    let sameAudio = false;

    if (result !== null) {
      oldList = JSON.parse(result);

      updatedList = oldList.filter((list) => {
        if (list.id === playList.id) {
          // Será verificado se o áudio selecionado já está dentro de uma
lista ou não
          for (let audio of list.audios) {
            if (audio.id === addToPlayList.id) {
              // Alerta comalguma mensagem
              sameAudio = true;
              return;
            }
          }
          // Caso contrário a playList será atualizada
          list.audios = [...list.audios, addToPlayList];
        }
        return list;
      });
    }
    if (sameAudio) {
      Alert.alert(
        "Foi encontrado o mesmo áudio!",
        `O áudio --> ${addToPlayList.filename} <-- já está dentro da
PlayList.`
      );
      sameAudio = false;
      return updateState(context, { addToPlayList: null });
    }
    updateState(context, { addToPlayList: null, playList: [...updatedList]
});
    return AsyncStorage.setItem("playlist",
JSON.stringify([...updatedList]));
  }

  // Se não for selecionado nenhum áudio, a lista pode ser aberta
  selectedPlayList = playList;
  navigation.navigate("PlayListDetail", playList);
 };

 return (
  <>
    <ScrollView contentContainerStyle={styles.container}>
      {playList.length
        ? playList.map((item) => (
            <TouchableOpacity
              key={item.id.toString()}
              style={styles.playListBanner}
              onPress={() => handleBannerPress(item)}
            >
              <Text style={styles.tituloModal}>{item.title}</Text>
              <Text style={styles.audioCount}>
                {item.audios.length > 1
                  ? `${item.audios.length} Músicas`
                  : `${item.audios.length} Música`}
              </Text>
            </TouchableOpacity>
          ))
```

```
         : null}

       <TouchableOpacity
         onPress={() => setModalVisible(true)}
         style={{ marginTop: 15 }}
       >
         <Text style={styles.playListBtn}>+ Nova PlayList</Text>
       </TouchableOpacity>

       <PlayListInputModal
         visible={modalVisible}
         onClose={() => setModalVisible(false)}
         onSubmit={createPlayList}
       />
     </ScrollView>
     <PlayListDetail
       visible={showPlayList}
       playList={selectedPlayList}
       onClose={() => setShowPlayList(false)}
     />
   </>
 );
};

// Estilos para o componente
const styles = StyleSheet.create({
 container: {
   padding: 20,
 },
 playListBanner: {
   padding: 5,
   backgroundColor: color.MODAL_LIST,
   borderRadius: 5,
   marginBottom: 15,
 },
 audioCount: {
   marginTop: 3,
   opacity: 0.5,
   fontSize: 14,
   color: color.FONT_MEDIUM,
 },
 playListBtn: {
   color: color.ACTIVE_BG,
   letterSpacing: 1,
   fontWeight: "bold",
   fontSize: 14,
   padding: 5,
 },
 tituloModal: {
   color: color.FONT_MEDIUM,
   fontWeight: "bold",
 },
});
```

```
export default PlayList;
```

PLAYLISTDETAIL.JS

```
import React, { useContext, useState } from "react";
import {
 View,
 StyleSheet,
 Modal,
 FlatList,
 Text,
 Dimensions,
 TouchableOpacity,
} from "react-native";
import color from "../misc/color";
import AudioListItem from "../components/AudioListItem";
import { selectAudio } from "../misc/audioController";
import { AudioContext } from "../context/AudioProvider";
import OptionModal from "../components/OptionModal";
import AsyncStorage from "@react-native-async-storage/async-storage";

const PlayListDetail = (props) => {
 const context = useContext(AudioContext);
 const playList = props.route.params;

 // Controle de visibilidade do modal, item selecionado e lista de áudios
 const [modalVisible, setModalVisible] = useState(false);
 const [selectedItem, setSelectedItem] = useState({});
 const [audios, setAudios] = useState(playList.audios);

 // Função para reproduzir um áudio
 const playAudio = async (audio) => {
   await selectAudio(audio, context, {
     activePlayList: playList,
     isPlayListRunning: true,
   });
 };

 // Função para fechar o modal
 const closeModal = () => {
   setSelectedItem({});
   setModalVisible(false);
 };

 // Função para remover um áudio da playlist
 const removeAudio = async () => {
   let isPlaying = context.isPlaying;
   let isPlayListRunning = context.isPlayListRunning;
   let soundObj = context.soundObj;
   let playbackPosition = context.playbackPosition;
   let activePlayList = context.activePlayList;
   //Parar o áudio, se o mesmo for removido da PlayList
   if (
     context.isPlayListRunning &&
     context.currentAudio.id === selectedItem.id
   ) {
     await context.playbackObj.stopAsync();
     await context.playbackObj.unloadAsync();

     isPlaying = false;
     isPlayListRunning = false;
     soundObj = null;
     playbackPosition = 0;
     activePlayList = [];
   }

   // Filtrar a lista de áudios para remover o áudio selecionado
   const newAudios = audios.filter((audio) => audio.id !==
selectedItem.id);

   // Atualizar a playlist
   const result = await AsyncStorage.getItem("playlist");

   if (result !== null) {
     const oldPlayLists = JSON.parse(result);
     const updatedPlayLists = oldPlayLists.filter((item) => {
       if (item.id === playList.id) {
         item.audios = newAudios;
       }

       return item;
     });

     AsyncStorage.setItem("playlist", JSON.stringify(updatedPlayLists));
     context.updateState(context, {
       playList: updatedPlayLists,
       isPlayListRunning,
       activePlayList,
       playbackPosition,
       isPlaying,
       soundObj,
     });
   }

   setAudios(newAudios);
   closeModal();
 };

 // Função para remover a playlist inteira
 const removePlaylist = async () => {
   let isPlaying = context.isPlaying;
   let isPlayListRunning = context.isPlayListRunning;
   let soundObj = context.soundObj;
```

```
    let playbackPosition = context.playbackPosition;
    let activePlayList = context.activePlayList;

    // Parar o áudio se a playlist estiver sendo reproduzida e for removida
    if (context.isPlayListRunning && activePlayList.id === playList.id) {
      await context.playbackObj.stopAsync();
      await context.playbackObj.unloadAsync();

      isPlaying = false;
      isPlayListRunning = false;
      soundObj = null;
      playbackPosition = 0;
      activePlayList = [];
    }

    // Atualizar a lista de playlists no AsyncStorage
    const result = await AsyncStorage.getItem("playlist");

    if (result !== null) {
      const oldPlayLists = JSON.parse(result);
      const updatedPlayLists = oldPlayLists.filter(
        (item) => item.id !== playList.id
      );

      AsyncStorage.setItem("playlist", JSON.stringify(updatedPlayLists));
      context.updateState(context, {
        playList: updatedPlayLists,
        isPlayListRunning,
        activePlayList,
        playbackPosition,
        isPlaying,
        soundObj,
      });
    }

    // Navegar de volta para a tela anterior
    props.navigation.goBack();
  };

  return (
    <>
      <View style={styles.container}>
        <View
          style={{
            width: "100%",
            flexDirection: "row",
            justifyContent: "space-between",
            paddingHorizontal: 15,
          }}
        >
          <Text style={styles.title}>{playList.title}</Text>
          <TouchableOpacity onPress={removePlaylist}>
            <Text style={[styles.title, { color: "darkred"
  }]}>Remover</Text>
          </TouchableOpacity>
        </View>

        {audios.length ? (
          <FlatList
            contentContainerStyle={styles.listContainer}
            data={audios}
            keyExtractor={(item) => item.id.toString()}
            renderItem={({ item }) => (
              <View style={{ marginBottom: 10 }}>
                <AudioListItem
                  title={item.filename}
                  duration={item.duration}
                  isPlaying={context.isPlaying}
                  activeListItem={item.id === context.currentAudio.id}
                  onAudioPress={() => playAudio(item)}
                  onOptionPress={() => {
                    setSelectedItem(item);
                    setModalVisible(true);
                  }}
                />
              </View>
            )}
          />
        ) : (
          <Text
            style={{
              fontWeight: "bold",
              color: color.FONT_LIGHT,
              fontSize: 25,
              paddingTop: 50,
            }}
          >
            Sem Áudio
          </Text>
        )}
      </View>
      <OptionModal
        visible={modalVisible}
        onClose={closeModal}
        options={[{ title: "Remover da PlayList", onPress: removeAudio }]}
        currentItem={selectedItem}
      />
    </>
  );
  };

  // Estilos para o componente
  const styles = StyleSheet.create({
  container: {
    alignItems: "center",
  },
  title: {
    textAlign: "center",
    fontSize: 20,
    paddingVertical: 5,
    fontWeight: "bold",
    color: color.ACTIVE_BG,
  },
  listContainer: {
    padding: 20,
  },
  });

  export default PlayListDetail;
```