import React, {useState, useEffect} from 'react';

import {Text, TouchableOpacity, View} from 'react-native';

import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';

import {AudioRecorder, AudioUtils} from 'react-native-audio';

import Sound from 'react-native-sound';

import {check, PERMISSIONS, request, RESULTS} from 'react-native-permissions';

import {CustomModal} from '../CustomModal'; // Ensure this path is correct

import Slider from '@react-native-community/slider';

function ChatAudioModel({bottomSheet, setBottomSheet}) {

const [isRecording, setIsRecording] = useState(false);

const [recordTime, setRecordTime] = useState(0);

const [hasRecorded, setHasRecorded] = useState(false);

const [sound, setSound] = useState(null);

const [isPlaying, setIsPlaying] = useState(false);

const [playbackTime, setPlaybackTime] = useState(0);

const audioPath = AudioUtils.DocumentDirectoryPath + '/test.aac';

const [sliderPosition, setSliderPosition] = useState(0);

const [showSoundEffectText, setShowSoundEffectText] = useState(false);

// Function to start recording

const startRecording = async () => {

const result = await check(PERMISSIONS.ANDROID.RECORD\_AUDIO);

if (result !== RESULTS.GRANTED) {

const requestResult = await request(PERMISSIONS.ANDROID.RECORD\_AUDIO);

if (requestResult !== RESULTS.GRANTED) {

return;

}

}

setRecordTime(0);

setIsRecording(true);

setHasRecorded(false);

try {

await AudioRecorder.prepareRecordingAtPath(audioPath, {

SampleRate: 22050,

Channels: 1,

AudioQuality: 'Low',

AudioEncoding: 'aac',

});

await AudioRecorder.startRecording();

} catch (error) {

console.error('Error starting recording: ', error);

}

};

// Function to stop recording

const stopRecording = async () => {

try {

await AudioRecorder.stopRecording();

setIsRecording(false);

setHasRecorded(true);

} catch (error) {

console.error('Error stopping recording: ', error);

}

};

// Playback the recorded audio

const playBackRecording = () => {

if (sound && isPlaying) {

// Pause playback

sound.pause();

setIsPlaying(false);

} else {

if (sound) {

// Resume playback

sound.play(() => {

sound.release();

setPlaybackTime(0);

setIsPlaying(false);

setSound(null); // Release sound instance after playback

});

setIsPlaying(true);

} else {

// First time playback

const newSound = new Sound(audioPath, '', error => {

if (error) {

console.log('Failed to load the sound', error);

return;

}

newSound.play(() => {

newSound.release();

setPlaybackTime(0);

setIsPlaying(false);

setSound(null); // Release sound instance after playback

});

});

setSound(newSound);

setIsPlaying(true);

}

}

};

useEffect(() => {

let interval;

if (isPlaying && sound) {

interval = setInterval(() => {

setPlaybackTime(prevTime => prevTime + 1);

sound.getCurrentTime(seconds => setSliderPosition(seconds));

}, 1000);

}

return () => clearInterval(interval);

}, [isPlaying, sound]);

// Cleanup sound object on component unmount

useEffect(() => {

return () => {

if (sound) {

sound.release();

}

};

}, [sound]);

const handlePlayPress = () => {

if (hasRecorded) {

playBackRecording();

}

};

const handleRecordPress = () => {

if (isRecording) {

stopRecording();

} else {

startRecording();

}

};

const handleSliderComplete = value => {

if (sound) {

sound.setCurrentTime(value);

setPlaybackTime(Math.floor(value));

}

};

// for model close

const handleClose = () => {

setBottomSheet(false);

};

return (

<CustomModal

borderBottomLeftRadius={0}

borderBottomRightRadius={0}

visible={bottomSheet}

modalHeight={400}

alignBottom={true}

backdropOpacity={0.2}

onBackdropPress={handleClose}

onModalHide={handleClose}>

<View

style={{

flex: 1,

flexDirection: 'column',

paddingHorizontal: 20,

marginVertical: 20,

}}>

<TouchableOpacity

onPress={handleClose}

style={{alignSelf: 'center', marginTop: 10}}>

<MaterialCommunityIcons

name="close-circle-outline"

size={22}

color="#E5214E"

/>

</TouchableOpacity>

<View

style={{

flexDirection: 'row',

justifyContent: 'center',

alignItems: 'center',

marginLeft: 'auto',

marginRight: 'auto',

//

marginTop:100

}}>

<TouchableOpacity

onPress={handlePlayPress}

style={{justifyContent: 'center', alignItems: 'center'}}>

<MaterialCommunityIcons

name={isPlaying ? 'pause' : 'play'}

size={30}

color={hasRecorded ? '#128C7E' : 'gray'} // Change color based on recording state

/>

</TouchableOpacity>

<Slider

style={{width: '70%', height: 40}}

minimumValue={0}

maximumValue={sound ? sound.getDuration() : 0}

value={sliderPosition}

onSlidingComplete={value => handleSliderComplete(value)}

minimumTrackTintColor={hasRecorded ? '#128C7E' : 'gray'}

maximumTrackTintColor="gray"

thumbTintColor={hasRecorded ? '#128C7E' : 'gray'}

/>

<View>

<Text style={{color: 'black'}}>{playbackTime} :00</Text>

</View>

</View>

{/\* <TouchableOpacity

onPress={handleRecordPress}

style={{

justifyContent: 'center',

alignContent: 'center',

marginLeft: 'auto',

marginRight: 'auto',

marginTop: 100,

}}>

<View

style={{

height: 60,

width: 60,

backgroundColor: '#128C7E',

borderRadius: 30,

justifyContent: 'center',

alignItems: 'center',

}}>

<MaterialCommunityIcons

name="microphone"

size={36}

color="#FFFFFF"

/>

</View>

</TouchableOpacity> \*/}

{/\* {isRecording && (

<View>

<Text style={{color: 'black'}}>

Recording Time: {recordTime} seconds

</Text>

</View>

)} \*/}

<View style={{ justifyContent: 'center', alignContent: 'center', marginTop: 100, flexDirection:"row" }}>

<TouchableOpacity style={{justifyContent:"center", alignItems:"center", marginLeft:"auto", marginRight:"auto"}}

onPressIn={startRecording}

onPressOut={stopRecording}

>

<View style={{ height: 60, width: 60, backgroundColor: '#128C7E', borderRadius: 30, justifyContent: 'center', alignItems: 'center' }}>

<MaterialCommunityIcons name="microphone" size={36} color='#FFFFFF' />

</View>

</TouchableOpacity>

<TouchableOpacity >

<MaterialCommunityIcons name="check" size={36} color='#128C7E' />

</TouchableOpacity>

</View>

</View>

</CustomModal>

);

}

export default ChatAudioModel;