React Native Audio Player Implementation Guide

Tech Stack

- TypeScript
- Expo
- Yarn
- React Native
- Expo Audio SDK

Architecture Overview

1. Core Audio Service Layer

Create a singleton service that wraps Expo Audio functionality:

- Handles all audio operations (play, pause, seek, etc.)
- Manages audio session configuration
- Provides a consistent API regardless of component lifecycle
- Handles background audio setup

2. State Management

Use a global state solution for audio player state:

- **Zustand** (lightweight, TypeScript-friendly) or **Redux Toolkit**
- Track: current track, playlist, playback status, position, duration
- Persist certain states (last played, position) using AsyncStorage

3. Component Structure

```
src/
├── services/
│   └── AudioService.ts          # Core audio logic
├── stores/
│   └── audioStore.ts            # Global state
├── hooks/
│   ├── useAudioPlayer.ts        # Main hook
│   └── usePlaybackStatus.ts     # Status updates
├── components/
│   ├── MiniPlayer.tsx           # Persistent mini player
│   ├── FullScreenPlayer.tsx     # Full player view
│   ├── PlaybackControls.tsx     # Reusable controls
│   └── ProgressBar.tsx          # Seek bar
└── types/
    └── audio.types.ts           # TypeScript definitions
```

Key Implementation Details

AudioService.ts Core Methods

typescript

```
class AudioService {
  private sound: Audio.Sound | null = null;
  private isInitialized = false;

  // Set up audio mode and event listeners
  async initialize(): Promise<void> {
    if (this.isInitialized) return;

    await Audio.setAudioModeAsync({
      staysActiveInBackground: true,
      playsInSilentModeIOS: true,
      interruptionModeIOS: InterruptionModeIOS.DuckOthers,
      interruptionModeAndroid: InterruptionModeAndroid.DuckOthers,
      shouldDuckAndroid: true,
    });

    this.isInitialized = true;
  }

  // Load audio from URI
  async loadTrack(track: Track): Promise<void> {
    // Unload previous sound if exists
    if (this.sound) {
      await this.sound.unloadAsync();
    }

    const { sound } = await Audio.Sound.createAsync(
      { uri: track.uri },
      { shouldPlay: false }
    );

    this.sound = sound;
  }

  async play(): Promise<void> {
    await this.sound?.playAsync();
  }

  async pause(): Promise<void> {
    await this.sound?.pauseAsync();
  }

  async seek(position: number): Promise<void> {
    await this.sound?.setPositionAsync(position);
  }

  // Playlist management
  setQueue(tracks: Track[]): void {
    // Implementation
  }

  async skipToNext(): Promise<void> {
    // Implementation
  }

  async skipToPrevious(): Promise<void> {
    // Implementation
  }
}

export default new AudioService();
```

State Management with Zustand

typescript

```typescript
// audioStore.ts
import { create } from 'zustand';
import { Track, PlaybackStatus } from '../types/audio.types';

interface AudioState {
  currentTrack: Track | null;
  queue: Track[];
  isPlaying: boolean;
  position: number;
  duration: number;
  isLoading: boolean;
  repeatMode: 'off' | 'all' | 'one';
  isShuffled: boolean;

  // Actions
  setCurrentTrack: (track: Track) => void;
  setQueue: (tracks: Track[]) => void;
  setPlaybackStatus: (status: PlaybackStatus) => void;
  setRepeatMode: (mode: 'off' | 'all' | 'one') => void;
  toggleShuffle: () => void;
}

export const useAudioStore = create<AudioState>((set) => ({
  currentTrack: null,
  queue: [],
  isPlaying: false,
  position: 0,
  duration: 0,
  isLoading: false,
  repeatMode: 'off',
  isShuffled: false,

  setCurrentTrack: (track) => set({ currentTrack: track }),
  setQueue: (tracks) => set({ queue: tracks }),
  setPlaybackStatus: (status) => set({
    isPlaying: status.isPlaying,
    position: status.positionMillis,
    duration: status.durationMillis,
    isLoading: status.isLoading,
  }),
  setRepeatMode: (mode) => set({ repeatMode: mode }),
  toggleShuffle: () => set((state) => ({ isShuffled: !state.isShuffled })),
}));
```

Custom Hook Implementation

typescript

```typescript
// useAudioPlayer.ts
import { useCallback, useEffect } from 'react';
import AudioService from '../services/AudioService';
import { useAudioStore } from '../stores/audioStore';

export const useAudioPlayer = () => {
  const {
    currentTrack,
    isPlaying,
    position,
    duration,
    queue,
    repeatMode,
    isShuffled
  } = useAudioStore();

  useEffect(() => {
    // Initialize audio service
    AudioService.initialize();
  }, []);

  const play = useCallback(async () => {
    await AudioService.play();
  }, []);

  const pause = useCallback(async () => {
    await AudioService.pause();
  }, []);

  const seek = useCallback(async (position: number) => {
    await AudioService.seek(position);
  }, []);

  const skipToNext = useCallback(async () => {
    await AudioService.skipToNext();
  }, []);

  const skipToPrevious = useCallback(async () => {
    await AudioService.skipToPrevious();
  }, []);

  const loadTrack = useCallback(async (track: Track) => {
    await AudioService.loadTrack(track);
    useAudioStore.getState().setCurrentTrack(track);
  }, []);

  return {
    // State
    currentTrack,
    isPlaying,
    position,
    duration,
    queue,
    repeatMode,
    isShuffled,

    // Actions
    play,
    pause,
    seek,
    skipToNext,
    skipToPrevious,
```

```
    loadTrack,
  };
};
```

Advanced Features

Queue Management

- Implement shuffle algorithm
- Preload next track for gapless playback
- Handle dynamic playlist updates
- Manage repeat modes (off, all, one)

Background Playback Configuration

typescript

```typescript
await Audio.setAudioModeAsync({
  staysActiveInBackground: true,
  playsInSilentModeIOS: true,
  interruptionModeIOS: InterruptionModeIOS.DuckOthers,
  interruptionModeAndroid: InterruptionModeAndroid.DuckOthers,
  shouldDuckAndroid: true,
  playThroughEarpieceAndroid: false,
});
```

Performance Optimizations

- Lazy load album artwork
- Use `React.memo` for control components
- Implement virtual scrolling for large playlists
- Cache loaded audio objects
- Debounce position updates to avoid excessive re-renders

Error Handling Strategy

typescript

```
class AudioService {
  async loadTrack(track: Track, retries = 3): Promise<void> {
    try {
      // Load logic
    } catch (error) {
      if (retries > 0) {
        // Exponential backoff
        await new Promise(resolve => setTimeout(resolve, 1000 * (4 - retries)));
        return this.loadTrack(track, retries - 1);
      }
      throw new AudioLoadError(`Failed to load track: ${track.title}`, error);
    }
  }

  private handleInterruption(interruption: InterruptionType): void {
    switch (interruption) {
      case 'phone_call':
        this.pause();
        break;
      case 'other_app':
        // Handle based on settings
        break;
    }
  }
}
```

UI/UX Considerations

Gestures

- Swipe down to minimize player
- Swipe on album art for skip
- Long press for additional options

Animations

Use `react-native-reanimated` for smooth transitions:

```typescript
const animatedStyle = useAnimatedStyle(() => ({
  transform: [{ translateY: withSpring(offset.value) }],
}));
```

Lock Screen Controls

Configure media controls for iOS/Android:

```typescript
// iOS: Use expo-av's Audio.setAudioModeAsync
// Android: Notification controls handled automatically
```

Accessibility

- Screen reader support with proper labels
- Minimum touch target size (44x44 points)
- High contrast mode support
- Keyboard navigation support

Testing Strategy

Unit Tests

```typescript
describe('AudioService', () => {
  it('should load a track successfully', async () => {
    const track = { id: '1', title: 'Test', url: 'test.mp3' };
    await AudioService.loadTrack(track);
    expect(AudioService.getCurrentTrack()).toEqual(track);
  });
});
```

Integration Tests

- Test store updates when playback status changes
- Test queue management operations
- Test background/foreground transitions

Mock Configuration

```typescript
jest.mock('expo-av', () => ({
  Audio: {
    Sound: {
      createAsync: jest.fn(),
    },
    setAudioModeAsync: jest.fn(),
  },
}));
```

Additional Libraries to Consider

Library	Purpose	When to Use
react-native-track-player	Advanced audio features	Need lockscreen controls, queue management
react-native-reanimated	Smooth animations	Complex UI transitions
react-native-gesture-handler	Better gesture handling	Custom gesture controls
expo-av	Video support	Adding video playback
expo-media-library	Device music access	Local music library integration
react-native-fast-image	Image caching	Album artwork optimization

Sample Component Implementation

MiniPlayer Component

```typescript
```

```typescript
import React from 'react';
import { View, Text, TouchableOpacity, Image } from 'react-native';
import { useAudioPlayer } from '../hooks/useAudioPlayer';

export const MiniPlayer: React.FC = () => {
  const { currentTrack, isPlaying, play, pause } = useAudioPlayer();

  if (!currentTrack) return null;

  return (
    <View style={styles.container}>
      <Image source={{ uri: currentTrack.artwork }} style={styles.artwork} />
      <View style={styles.info}>
        <Text style={styles.title}>{currentTrack.title}</Text>
        <Text style={styles.artist}>{currentTrack.artist}</Text>
      </View>
      <TouchableOpacity onPress={isPlaying ? pause : play}>
        <Icon name={isPlaying ? 'pause' : 'play'} />
      </TouchableOpacity>
    </View>
  );
};
```

ProgressBar Component

```typescript
import React from 'react';
import { View } from 'react-native';
import Slider from '@react-native-community/slider';
import { useAudioPlayer } from '../hooks/useAudioPlayer';

export const ProgressBar: React.FC = () => {
  const { position, duration, seek } = useAudioPlayer();

  return (
    <Slider
      value={position}
      minimumValue={0}
      maximumValue={duration}
      onSlidingComplete={seek}
      minimumTrackTintColor="#1976D2"
      maximumTrackTintColor="#BBBBBB"
    />
  );
};
```

TypeScript Type Definitions

```typescript
```

```typescript
// audio.types.ts
export interface Track {
  id: string;
  url: string;
  title: string;
  artist: string;
  album?: string;
  artwork?: string;
  duration?: number;
}

export interface PlaybackStatus {
  isLoaded: boolean;
  isPlaying: boolean;
  isBuffering: boolean;
  durationMillis: number;
  positionMillis: number;
  rate: number;
  shouldPlay: boolean;
  volume: number;
  isMuted: boolean;
  isLooping: boolean;
  didJustFinish: boolean;
}

export interface AudioPlayerState {
  currentTrack: Track | null;
  queue: Track[];
  history: Track[];
  isPlaying: boolean;
  position: number;
  duration: number;
  isLoading: boolean;
  error: Error | null;
}
```

Getting Started

1. **Install dependencies:**

```bash
yarn add expo-av zustand
yarn add -D @types/react-native
```

2. **Set up audio permissions (app.json):**

```json
{
  "expo": {
    "plugins": [
      [
        "expo-av",
        {
          "microphonePermission": "Allow $(PRODUCT_NAME) to access your microphone."
        }
      ]
    ]
  }
}
```

3. **Initialize the audio service in your App.tsx:**

```typescript
import AudioService from './src/services/AudioService';

export default function App() {
  useEffect(() => {
    AudioService.initialize();
  }, []);

  return <YourAppContent />;
}
```

4. **Implement the mini player in your main navigation:**

```typescript
<View style={{ flex: 1 }}>
  <NavigationContainer>
    {/* Your screens */}
  </NavigationContainer>
  <MiniPlayer />
</View>
```

Best Practices

1. **Always handle async operations with try-catch**
2. **Unload audio resources when component unmounts**
3. **Implement proper loading states**
4. **Cache frequently accessed data**
5. **Use TypeScript strictly for better type safety**
6. **Follow React Native performance guidelines**
7. **Test on both iOS and Android devices**
8. **Implement analytics for playback events**
9. **Handle network connectivity changes**
10. **Provide offline playback capabilities**

Troubleshooting Common Issues

Issue	Solution
Audio stops in background	Ensure `staysActiveInBackground: true` is set
Delayed playback start	Preload audio when possible
Memory leaks	Always unload audio objects when done
Interruption handling	Implement proper interruption observers
Silent mode issues (iOS)	Set `playsInSilentModeIOS: true`

Resources

- Expo Audio Documentation
- React Native Performance
- Zustand Documentation
- TypeScript Best Practices