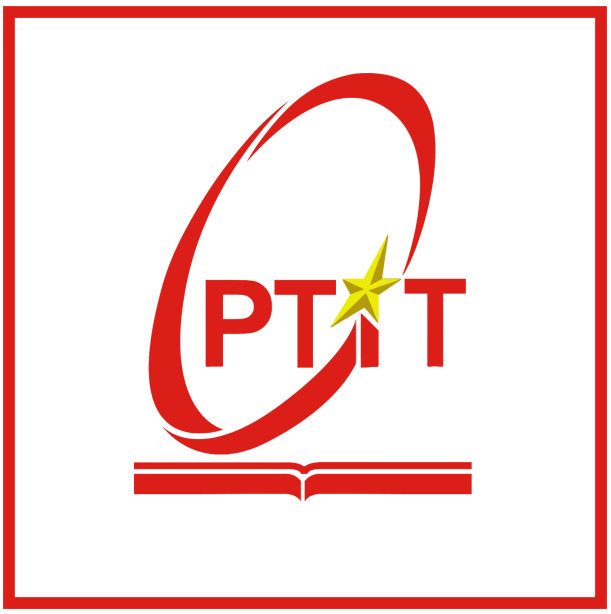
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 10

TRIỂN KHAI
UX/UI VÀ FRONT END, BACK END
(TIẾP)

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

- Nguyễn Quang Trung - B22DCDT321

MỤC LỤC

Stores

Store đóng vai trò là nơi quản lý state (trạng thái) của ứng dụng bằng cách sử dụng thư viện Zustand. Đây là một pattern quản lý state phổ biến trong React, tương tự như Redux nhưng đơn giản hơn.

Zustand là một thư viện quản lý trạng thái đơn giản và nhẹ. Khi bạn muốn thực hiện các thao tác bất đồng bộ như gọi API trong Zustand, bạn viết một **hàm async (trả về Promise)** bên trong action.

Nó có thể ở 1 trong 3 trạng thái:

1. **Pending (đang chờ):** chưa hoàn thành.
2. **Fulfilled (hoàn thành):** thành công.
3. **Rejected (thất bại):** có lỗi.

Cụ thể:

- useAuthStore:

```ts
import { axiosInstance } from "@/lib/axios";
import { create } from "zustand";

interface AuthStore {
  isAdmin: boolean;
  isLoading: boolean;
  error: string | null;

  checkAdminStatus: () => Promise<void>;
  reset: () => void;
}

export const useAuthStore = create<AuthStore>((set) => ({
  isAdmin: false,
  isLoading: false,
  error: null,

  checkAdminStatus: async () => {
    set({ isLoading: true, error: null });
    try {
      const response = await axiosInstance.get("/admin/check");
      set({ isAdmin: response.data.admin });
    } catch (error: any) {
      set({ isAdmin: false, error: error.response.data.message });
    } finally {
      set({ isLoading: false });
    }
  },

  reset: () => {
    set({ isAdmin: false, isLoading: false, error: null });
  },
}));
```

- Quản lý trạng thái xác thực người dùng
- Theo dõi trạng thái admin (isAdmin)
- Xử lý việc kiểm tra quyền admin
- Quản lý trạng thái loading và lỗi

→ Giải thích chi tiết useAuthStore:

1. Import statements:

- Import axiosInstance từ thư viện axios đã được cấu hình sẵn để gọi API
- Import hàm create từ thư viện Zustand để tạo store

2. Interface định nghĩa:
    - Định nghĩa kiểu dữ liệu cho store với các thuộc tính:
        - isAdmin: boolean để kiểm tra người dùng có phải admin không
        - isLoading: boolean để theo dõi trạng thái loading
        - error: string hoặc null để lưu thông báo lỗi
        - checkAdminStatus: hàm bất đồng bộ để kiểm tra quyền admin
        - reset: hàm để reset lại trạng thái store
3. Tạo store:
    - Sử dụng create từ Zustand để tạo store
    - Khởi tạo các giá trị mặc định cho state
4. Hàm checkAdminStatus:
    - Hàm này được sử dụng để kiểm tra quyền admin
    - Gọi API /admin/check thông qua axiosInstance
    - Xử lý các trạng thái loading và error
    - Cập nhật state isAdmin dựa trên response từ server
5. Hàm reset: Reset tất cả giá trị về mặc định

→ useAuthStore được gọi trong lớp AuthProvider mỗi khi client truy cập vào trang web. Tại đây, useAuthStore sẽ được cập nhật các giá trị tương ứng.

- useMusicStore:

useMusicStore.ts

frontend > src > stores > useMusicStore.ts > useMusicStore > create() callback > deleteSong

```ts
import { axiosInstance } from "@/lib/axios";
import { Album, Song, Stats } from "@/types";
import toast from "react-hot-toast";
import { create } from "zustand";

interface MusicStore {
    songs: Song[];
    albums: Album[];
    isLoading: boolean;
    error: string | null;
    currentAlbum: Album | null;
    featuredSongs: Song[];
    madeForYouSongs: Song[];
    trendingSongs: Song[];
    stats: Stats;

    fetchAlbums: () => Promise<void>;
    fetchAlbumById: (id: string) => Promise<void>;
    fetchFeaturedSongs: () => Promise<void>;
    fetchMadeForYouSongs: () => Promise<void>;
    fetchTrendingSongs: () => Promise<void>;
    fetchStats: () => Promise<void>;
    fetchSongs: () => Promise<void>;
    deleteSong: (id: string) => Promise<void>;
    deleteAlbum: (id: string) => Promise<void>;
}

export const useMusicStore = create<MusicStore>((set) => ({
    albums: [],
    songs: [],
    isLoading: false,
    error: null,
    currentAlbum: null,
    madeForYouSongs: [],
    featuredSongs: [],
    trendingSongs: [],
    stats: {
        totalSongs: 0,
        totalAlbums: 0,
        totalUsers: 0,
        totalArtists: 0,
    },

    deleteSong: async (id) => {…
    },

    deleteAlbum: async (id) => {…
    },
```

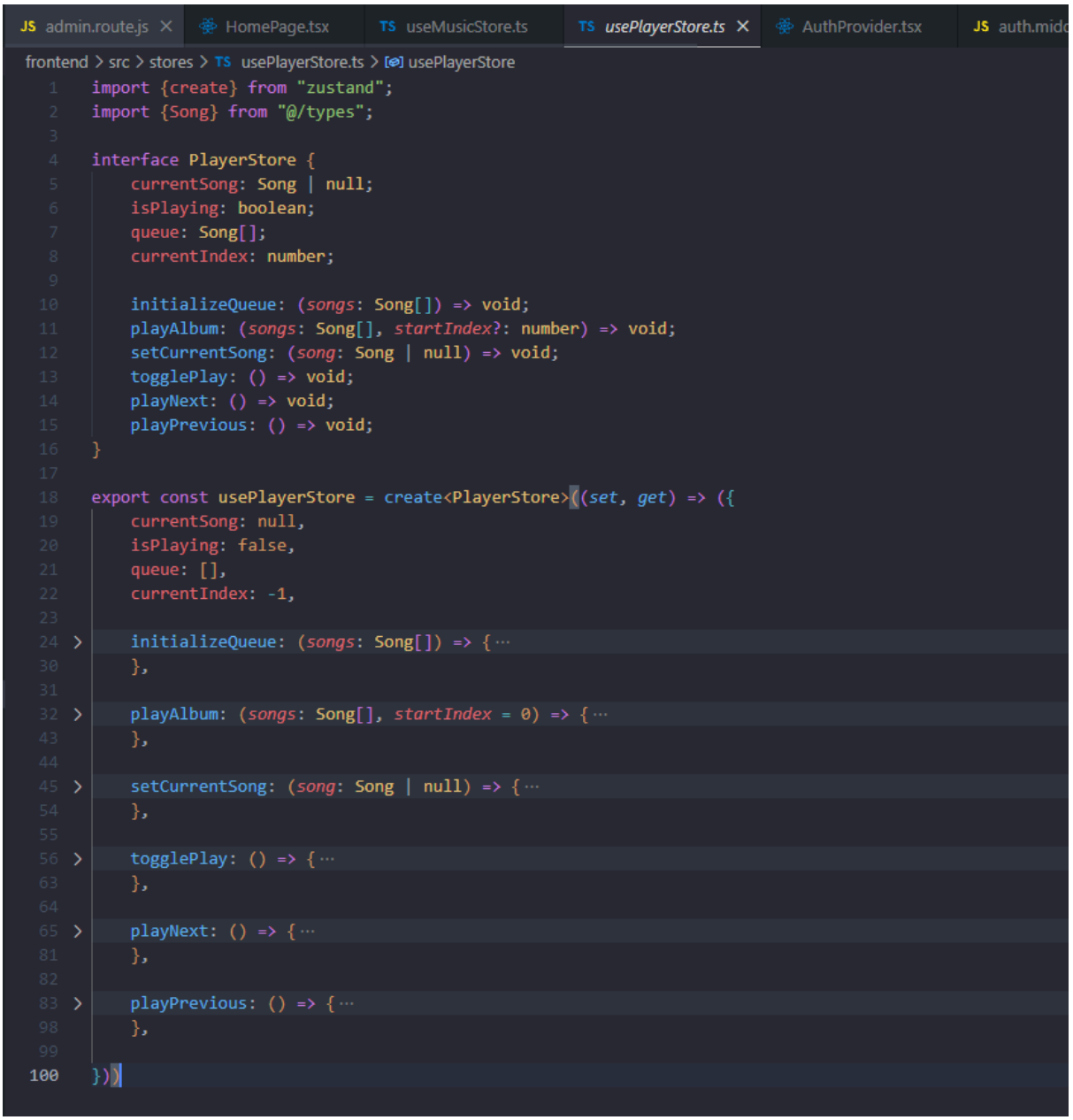
```ts
export const useMusicStore = create<MusicStore>((set) => ({
    albums: [],
    songs: [],
    isLoading: false,
    error: null,
    currentAlbum: null,
    madeForYouSongs: [],
    featuredSongs: [],
    trendingSongs: [],
    stats: {
        totalSongs: 0,
        totalAlbums: 0,
        totalUsers: 0,
        totalArtists: 0,
    },

    deleteSong: async (id) => {…
    },

    deleteAlbum: async (id) => {…
    },

    fetchSongs: async () => {…
    },

    fetchStats: async () => {…
    },

    fetchAlbums: async () => {…
    },

    fetchAlbumById: async (id) => {…
    },

    fetchFeaturedSongs: async () => {…
    },

    fetchMadeForYouSongs: async () => {…
    },

    fetchTrendingSongs: async () => {…
    },
}));
```

- Quản lý dữ liệu âm nhạc như songs, albums
- Xử lý các thao tác CRUD với bài hát và album
- Quản lý các danh sách bài hát đặc biệt (featured, trending, made for you)
- Theo dõi thống kê (stats) của hệ thống

→ Luồng hoạt động: Khi component mount, gọi các hàm fetch để lấy dữ liệu.

- Trong quá trình fetch:
  - isLoading = true
  - Hiển thị loading UI
- Sau khi fetch xong:
  - Cập nhật state tương ứng
  - isLoading = false
  - Render UI với dữ liệu mới
- Nếu có lỗi:
  - Cập nhật error
  - Hiển thị thông báo lỗi

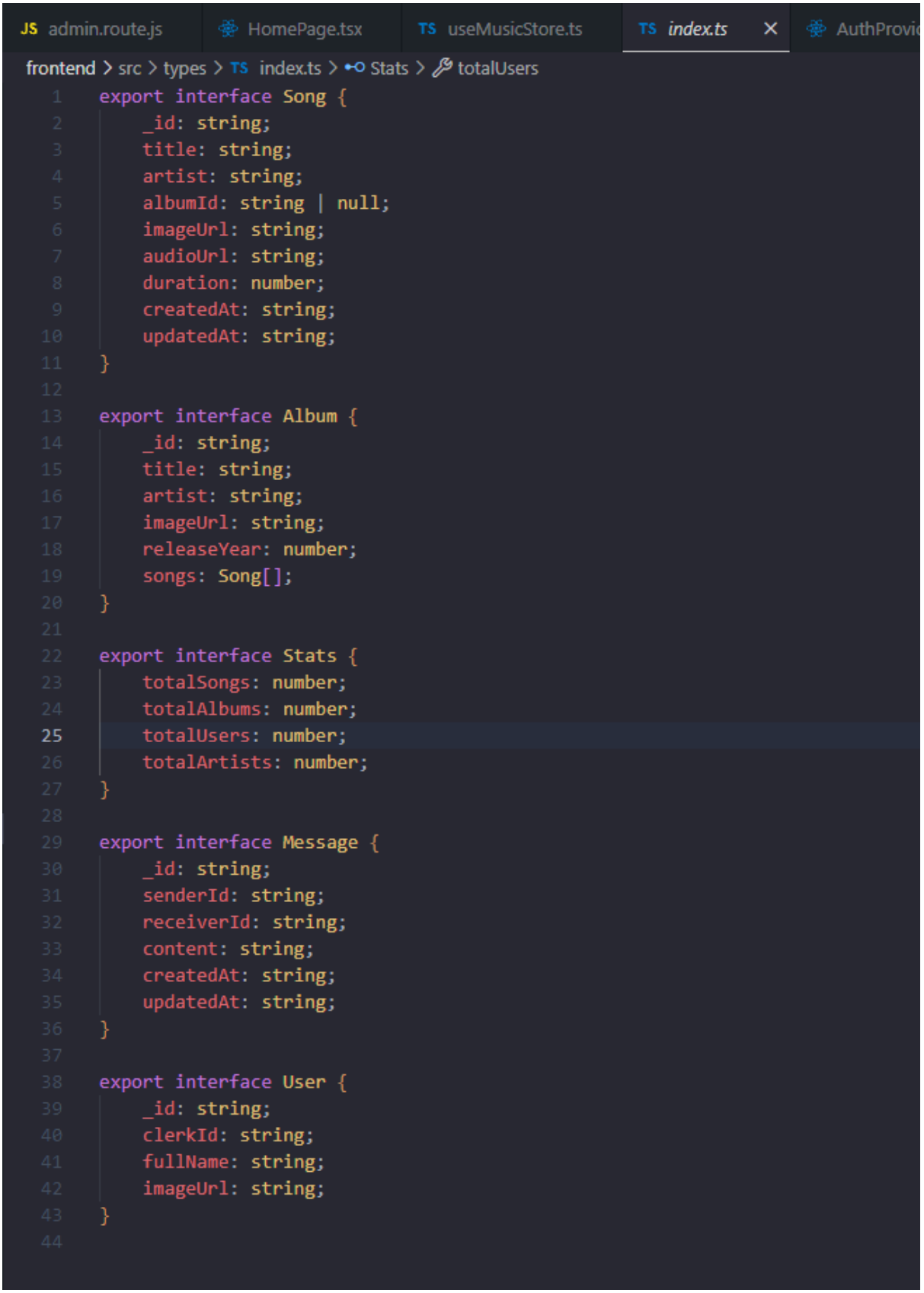
- usePlayerStore:



→ Tác dụng:

- Quản lý trạng thái phát nhạc
- Đồng bộ trạng thái phát nhạc
- Quản lý queue phát nhạc
- Xử lý logic phức tạp

Types


admin.route.js
HomePage.tsx
useMusicStore.ts
index.ts
AuthProvi
frontend > src > types > index.ts > Stats > totalUsers
export interface Song {
_id: string;
title: string;
artist: string;
albumId: string | null;
imageUrl: string;
audioUrl: string;
duration: number;
createdAt: string;
updatedAt: string;
}
export interface Album {
_id: string;
title: string;
artist: string;
imageUrl: string;
releaseYear: number;
songs: Song[];
}
export interface Stats {
totalSongs: number;
totalAlbums: number;
totalUsers: number;
totalArtists: number;
}
export interface Message {
_id: string;
senderId: string;
receiverId: string;
content: string;
createdAt: string;
updatedAt: string;
}
export interface User {
_id: string;
clerkId: string;
fullName: string;
imageUrl: string;
}

Type giúp định nghĩa kiểu dữ liệu (type definitions)

- Định nghĩa cấu trúc dữ liệu cho bài hát
- Giúp TypeScript kiểm tra kiểu dữ liệu
- Tự động gợi ý code (autocomplete) khi sử dụng

→ Các component khác có thể import và sử dụng

→ Lợi ích: type safety, code completion.