这是一个可以正确运行的聊天室全栈web项目，基于Node.js, 使用了Next.js(包括本地API Routes), React, Prisma。给出D:/jychatroom/chatroom/src目录下主要文件代码如下：

**chatroom/src/hooks/useAuth.ts：**

import { useState, useEffect } from 'react';

import jwt from 'jsonwebtoken';

interface DecodedToken {

  sub: string;

  username: string;

  exp: number;

}

export function useAuth() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const [username, setUsername] = useState<string | null>(null);

  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {

    checkAuthStatus();

  }, []);

  const checkAuthStatus = () => {

    try {

      const token = localStorage.getItem('token');

      if (!token) {

        setIsLoggedIn(false);

        setUsername(null);

        setIsLoading(false);

        return;

      }

      // 解码 token 检查是否过期

      const decoded = jwt.decode(token) as DecodedToken;

      if (!decoded || !decoded.exp) {

        // 无效的 token

        localStorage.removeItem('token');

        setIsLoggedIn(false);

        setUsername(null);

        setIsLoading(false);

        return;

      }

      // 检查是否过期

      const currentTime = Math.floor(Date.now() / 1000);

      if (decoded.exp < currentTime) {

        // token 过期

        localStorage.removeItem('token');

        setIsLoggedIn(false);

        setUsername(null);

        setIsLoading(false);

        return;

      }

      // token 有效

      setIsLoggedIn(true);

      setUsername(decoded.username);

      setIsLoading(false);

    } catch (error) {

      console.error('检查登录状态失败:', error);

      localStorage.removeItem('token');

      setIsLoggedIn(false);

      setUsername(null);

      setIsLoading(false);

    }

  };

  const logout = () => {

    localStorage.removeItem('token');

    localStorage.removeItem('nickname');

    setIsLoggedIn(false);

    setUsername(null);

  };

  const validateToken = () => {

    try {

      const token = localStorage.getItem('token');

      if (!token) {

        return false;

      }

      const decoded = jwt.decode(token) as DecodedToken;

      if (!decoded || !decoded.exp) {

        return false;

      }

      const currentTime = Math.floor(Date.now() / 1000);

      return decoded.exp >= currentTime;

    } catch (error) {

      console.error('Token 验证失败:', error);

      return false;

    }

  };

  return {

    isLoggedIn,

    username,

    isLoading,

    checkAuthStatus,

    logout,

    validateToken

  };

}

**chatroom/src/lib/fetcher.ts：**

// 使用本地API

const prefix = ''; // 空字符串，使用相对路径

export async function authFetch(

  path: string,

  init: RequestInit = {}

) {

  const token = typeof window !== 'undefined' ? localStorage.getItem('token') : null;

  // 构造 headers 对象

  const headers: Record<string, string> = {

    'Content-Type': 'application/json',

    ...Object(init.headers || {}),

  };

  // 确保 token 存在时才添加 Authorization header

  if (token) {

    headers.Authorization = `Bearer ${token}`;

  }

  try {

    const response = await fetch(prefix + path, {

      ...init,

      headers,

    });

    if (!response.ok) {

      const errorData = await response.json().catch(() => ({}));

      throw new Error(errorData.message || `HTTP error! status: ${response.status}`);

    }

    return response;

  } catch (error) {

    console.error('Fetch error:', error);

    throw error;

  }

}

**src/middleware.ts：**

import { NextResponse } from 'next/server';

import type { NextRequest } from 'next/server';

// middleware.ts - 简化版本，因为jwt需要在API路由内部验证

export function middleware(req: NextRequest) {

  // 中间件不验证JWT（因为Edge Runtime不支持jsonwebtoken包）

  // JWT验证移到各个API路由内部处理

  return NextResponse.next();

}

export const config = {

  matcher: '/api/:path\*'

}

**chatroom/src/app/api/auth/login/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import bcrypt from 'bcryptjs';

import jwt from 'jsonwebtoken';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

const JWT\_SECRET = process.env.JWT\_SECRET || 'dev\_secret';

export async function POST(req: NextRequest) {

  try {

    const { username, password } = await req.json();

    const user = await prisma.user.findUnique({ where: { username } });

    if (!user || !(await bcrypt.compare(password, user.password))) {

      return NextResponse.json({ code: 1, message: '用户名或密码错误' }, { status: 401 });

    }

    const token = jwt.sign({ sub: user.id, username: user.username }, JWT\_SECRET, { expiresIn: '7d' });

    return NextResponse.json({ code: 0, data: { accessToken: token } });

  } catch {

    return NextResponse.json({ code: 1, message: '服务器错误' }, { status: 500 });

  }

}

**chatroom/src/app/api/auth/signup/route.ts:**

import { NextRequest, NextResponse } from 'next/server';

import bcrypt from 'bcryptjs';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function POST(req: NextRequest) {

  try {

    const { username, password } = await req.json();

    if (!username || !password)

      return NextResponse.json({ code: 1, message: '参数缺失' }, { status: 400 });

    const exists = await prisma.user.findUnique({ where: { username } });

    if (exists)

      return NextResponse.json({ code: 1, message: '用户名已存在' }, { status: 400 });

    const hashed = await bcrypt.hash(password, 10);

    await prisma.user.create({ data: { username, password: hashed } });

    return NextResponse.json({ code: 0, message: '注册成功' });

  } catch {

    return NextResponse.json({ code: 1, message: '服务器错误' }, { status: 500 });

  }

}

**chatroom/src/app/api/message/add/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function POST(req: NextRequest) {

  try {

    const { roomId, content, sender } = await req.json();

    if (!roomId || !content || !sender) {

      return NextResponse.json({

        code: 1,

        message: '参数缺失'

      }, { status: 400 });

    }

    const message = await prisma.message.create({

      data: {

        roomId: Number(roomId),

        content,

        sender

      },

    });

    return NextResponse.json({

      code: 0,

      data: null,

      message: '发送成功'

    });

  } catch (error) {

    console.error('发送消息失败:', error);

    return NextResponse.json({

      code: 1,

      message: '发送消息失败'

    }, { status: 500 });

  }

}

**chatroom/src/app/api/message/getUpdate/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET(req: NextRequest) {

  try {

    const { searchParams } = new URL(req.url);

    const roomId = searchParams.get('roomId');

    const sinceMessageId = searchParams.get('sinceMessageId');

    if (!roomId || !sinceMessageId) {

      return NextResponse.json({ code: 1, message: '缺少参数' }, { status: 400 });

    }

    const list = await prisma.message.findMany({

      where: { roomId: Number(roomId), id: { gt: Number(sinceMessageId) } },

      orderBy: { createdAt: 'asc' },

    });

    const messages = list.map((m) => ({

      messageId: m.id,

      roomId: m.roomId,

      sender: m.sender,

      content: m.content,

      time: m.createdAt.getTime(),

    }));

    return NextResponse.json({ code: 0, data: { messages } });

  } catch (e) {

    console.error('增量拉取失败:', e);

    return NextResponse.json({ code: 1, message: '服务器错误' }, { status: 500 });

  }

}

**chatroom/src/app/api/message/list/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET(req: NextRequest) {

  try {

    const { searchParams } = new URL(req.url);

    const roomId = searchParams.get('roomId');

    if (!roomId) {

      return NextResponse.json({

        code: 1,

        message: '缺少 roomId 参数'

      }, { status: 400 });

    }

    const messages = await prisma.message.findMany({

      where: { roomId: Number(roomId) },

      orderBy: { createdAt: 'asc' },

    });

    const formattedMessages = messages.map(msg => ({

      messageId: msg.id,

      roomId: msg.roomId,

      sender: msg.sender,

      content: msg.content,

      time: msg.createdAt.getTime(),

    }));

    return NextResponse.json({

      code: 0,

      data: { messages: formattedMessages }

    });

  } catch (error) {

    console.error('获取消息列表失败:', error);

    return NextResponse.json({

      code: 1,

      message: '获取消息失败'

    }, { status: 500 });

  }

}

**chatroom/src/app/api/room/delete/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

import jwt from 'jsonwebtoken';

const prisma = new PrismaClient();

const JWT\_SECRET = process.env.JWT\_SECRET || 'dev\_secret';

export async function POST(req: NextRequest) {

  try {

    const authHeader = req.headers.get('authorization');

    if (!authHeader?.startsWith('Bearer ')) {

      return NextResponse.json({ code: 1, message: '未登录' }, { status: 401 });

    }

    const token = authHeader.slice(7);

    try {

      const decoded = jwt.verify(token, JWT\_SECRET) as any;

      console.log('删除操作用户:', decoded.username);

    } catch (error) {

      return NextResponse.json({ code: 1, message: 'token 无效' }, { status: 401 });

    }

    const { roomId } = await req.json();

    if (!roomId) {

      return NextResponse.json({ code: 1, message: '缺少 roomId' }, { status: 400 });

    }

    console.log('删除房间 ID:', roomId);

    // 先检查房间是否存在

    const room = await prisma.room.findUnique({

      where: { id: Number(roomId) }

    });

    if (!room) {

      return NextResponse.json({ code: 1, message: '房间不存在' }, { status: 404 });

    }

    // 删除房间内的所有消息

    await prisma.message.deleteMany({

      where: { roomId: Number(roomId) }

    });

    // 删除房间

    await prisma.room.delete({

      where: { id: Number(roomId) }

    });

    console.log('房间删除成功');

    return NextResponse.json({ code: 0, message: '删除成功' });

  } catch (error) {

    console.error('删除房间错误:', error);

    return NextResponse.json({ code: 1, message: '删除失败' }, { status: 500 });

  }

}

**chatroom/src/app/api/room/add/route.ts:**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

import jwt from 'jsonwebtoken';

const prisma = new PrismaClient();

const JWT\_SECRET = process.env.JWT\_SECRET || 'dev\_secret';

export async function POST(req: NextRequest) {

  const token = req.headers.get('authorization')?.split(' ')[1];

  if (!token) return NextResponse.json({ code: 1, message: '未登录' }, { status: 401 });

  try {

    // 验证 token

    jwt.verify(token, JWT\_SECRET);

    const { roomName } = await req.json();

    if (!roomName || typeof roomName !== 'string' || roomName.trim() === '') {

      return NextResponse.json({ code: 1, message: '房间名不能为空' }, { status: 400 });

    }

    console.log(`正在创建房间: ${roomName.trim()}`);

    const newRoom = await prisma.room.create({ data: { name: roomName.trim() } });

    console.log(`房间创建成功, ID: ${newRoom.id}`);

    return NextResponse.json({ code: 0, data: { roomId: newRoom.id } });

  } catch (error) {

    // <-- 修正：捕获并记录详细错误信息

    console.error('创建房间失败:', error);

    const message = error instanceof Error ? error.message : '未知服务器错误';

    return NextResponse.json({ code: 1, message: `创建房间失败: ${message}` }, { status: 500 });

  }

}

**chatroom/src/app/api/room/list/route.ts**

import { NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET() {

  // 先拿房间

  const rooms = await prisma.room.findMany({ orderBy: { id: 'asc' } });

  // 再补最后一条消息

  const data = await Promise.all(

    rooms.map(async (r) => {

      const lastMsg = await prisma.message.findFirst({

        where: { roomId: r.id },

        orderBy: { createdAt: 'desc' },

      });

      return {

        roomId: r.id,

        roomName: r.name,

        lastMessage: lastMsg

          ? {

              messageId: lastMsg.id,

              roomId: lastMsg.roomId,

              sender: lastMsg.sender,

              content: lastMsg.content,

              time: lastMsg.createdAt.getTime(),

            }

          : null,

      };

    })

  );

  return NextResponse.json({ code: 0, data: { rooms: data } });

}

**chatroom/src/app/api/room/message/getUpdate/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET(req: NextRequest) {

  try {

    const { searchParams } = new URL(req.url);

    const roomId = searchParams.get('roomId');

    const sinceMessageId = searchParams.get('sinceMessageId');

    if (!roomId) {

      return NextResponse.json({

        code: 1,

        message: '缺少 roomId 参数'

      }, { status: 400 });

    }

    if (!sinceMessageId) {

      return NextResponse.json({

        code: 1,

        message: '缺少 sinceMessageId 参数'

      }, { status: 400 });

    }

    // 获取指定消息ID之后的所有新消息

    const messages = await prisma.message.findMany({

      where: {

        roomId: Number(roomId),

        id: {

          gt: Number(sinceMessageId) // greater than - 大于指定的消息ID

        }

      },

      orderBy: { createdAt: 'asc' },

    });

    const formattedMessages = messages.map(msg => ({

      messageId: msg.id,

      roomId: msg.roomId,

      sender: msg.sender,

      content: msg.content,

      time: msg.createdAt.getTime(),

    }));

    return NextResponse.json({

      code: 0,

      data: { messages: formattedMessages }

    });

  } catch (error) {

    console.error('获取增量消息失败:', error);

    return NextResponse.json({

      code: 1,

      message: '获取消息失败'

    }, { status: 500 });

  }

}

**chatroom/src/app/api/room/message/list/route.ts：**

import { NextRequest, NextResponse } from 'next/server';

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET(req: NextRequest) {

  try {

    const { searchParams } = new URL(req.url);

    const roomId = searchParams.get('roomId');

    if (!roomId) {

      return NextResponse.json({

        code: 1,

        message: '缺少 roomId 参数'

      }, { status: 400 });

    }

    const messages = await prisma.message.findMany({

      where: { roomId: Number(roomId) },

      orderBy: { createdAt: 'asc' },

    });

    const formattedMessages = messages.map(msg => ({

      messageId: msg.id,

      roomId: msg.roomId,

      sender: msg.sender,

      content: msg.content,

      time: msg.createdAt.getTime(),

    }));

    return NextResponse.json({

      code: 0,

      data: { messages: formattedMessages }

    });

  } catch (error) {

    console.error('获取消息列表失败:', error);

    return NextResponse.json({

      code: 1,

      message: '获取消息失败'

    }, { status: 500 });

  }

}

**chatroom/src/app/chat/page.tsx：**

'use client';

import { useEffect, useState } from 'react';

import { useRouter } from 'next/navigation';

import useSWR from 'swr';

import useSWRMutation from 'swr/mutation';

import styles from './page.module.css';

import { authFetch } from '@/lib/fetcher';

import { useAuth } from '@/hooks/useAuth';

// 类型定义

interface Message {

  messageId: number;

  roomId: number;

  sender: string;

  content: string;

  time: number;

}

interface RoomPreviewInfo {

  roomId: number;

  roomName: string;

  lastMessage: Message | null;

}

interface RoomListRes {

  rooms: RoomPreviewInfo[];

}

interface RoomAddArgs {

  user: string;

  roomName: string;

}

interface RoomAddRes {

  roomId: number;

}

interface RoomMessageListRes {

  messages: Message[];

}

interface MessageAddArgs {

  roomId: number;

  content: string;

  sender: string;

}

// 单条消息组件

interface MessageItemProps {

  message: Message;

  currentUser: string;

}

function MessageItem({ message, currentUser }: MessageItemProps) {

  const isOwn = message.sender === currentUser;

  // 格式化时间为 "年-月-日 时:分" 格式

  const formatTime = (timestamp: number) => {

    const date = new Date(timestamp);

    const year = date.getFullYear();

    const month = String(date.getMonth() + 1).padStart(2, '0');

    const day = String(date.getDate()).padStart(2, '0');

    const hours = String(date.getHours()).padStart(2, '0');

    const minutes = String(date.getMinutes()).padStart(2, '0');

    return `${year}-${month}-${day} ${hours}:${minutes}`;

  };

  return (

    <div className={`${styles.messageItem} ${isOwn ? styles.ownMessage : styles.otherMessage}`}>

      <div className={styles.messageHeader}>

        <span className={styles.sender}>{message.sender}</span>

        <span className={styles.time}>{formatTime(message.time)}</span>

      </div>

      <div className={styles.messageContent}>{message.content}</div>

    </div>

  );

}

// 登录状态组件

function AuthStatus({ isLoggedIn, username, onLogout }: {

  isLoggedIn: boolean;

  username: string | null;

  onLogout: () => void;

}) {

  return (

    <div className={styles.authStatus}>

      {isLoggedIn ? (

        <div className={styles.loggedIn}>

          <span className={styles.userInfo}>已登录: {username}</span>

          <button

            className={styles.logoutBtn}

            onClick={onLogout}

            title="退出登录"

          >

            退出

          </button>

        </div>

      ) : (

        <div className={styles.notLoggedIn}>

          <span className={styles.guestInfo}>游客模式</span>

          <a href="/login" className={styles.loginLink}>

            去登录

          </a>

        </div>

      )}

    </div>

  );

}

// 权限提示组件

function PermissionAlert({ message, onClose }: { message: string; onClose: () => void }) {

  return (

    <div className={styles.permissionAlert}>

      <div className={styles.alertContent}>

        <p>{message}</p>

        <div className={styles.alertButtons}>

          <button onClick={onClose} className={styles.alertCloseBtn}>

            知道了

          </button>

          <a href="/login" className={styles.alertLoginBtn}>

            去登录

          </a>

        </div>

      </div>

    </div>

  );

}

// 单个房间组件

interface RoomEntryProps {

  room: RoomPreviewInfo;

  onClick: () => void;

  isActive: boolean;

  onDelete: () => void;

  canDelete: boolean;

}

function RoomEntry({ room, onClick, isActive, onDelete, canDelete }: RoomEntryProps) {

  const handleDeleteClick = (e: React.MouseEvent) => {

    e.stopPropagation();

    onDelete();

  };

  return (

    <div

      className={`${styles.roomItem} ${isActive ? styles.active : ''}`}

      onClick={onClick}

    >

      <h3 className={styles.roomName}>{room.roomName}</h3>

      <p className={styles.lastMessage}>

        {room.lastMessage

          ? `${room.lastMessage.sender}: ${room.lastMessage.content}`

          : '暂无消息'}

      </p>

      {canDelete && (

        <button

          className={styles.deleteBtn}

          onClick={handleDeleteClick}

          title="删除房间"

        >

          ×

        </button>

      )}

      {!canDelete && (

        <div className={styles.noPermissionHint} title="需要登录才能删除房间">

          {/\* 可以添加一个禁用状态的图标 \*/}

        </div>

      )}

    </div>

  );

}

export default function ChatPage() {

  const [nickname, setNickname] = useState('');

  const [selectedRoomId, setSelectedRoomId] = useState<number | null>(null);

  const [showAddModal, setShowAddModal] = useState(false);

  const [newRoomName, setNewRoomName] = useState('');

  const [messageInput, setMessageInput] = useState('');

  const [showPermissionAlert, setShowPermissionAlert] = useState(false);

  const [permissionAlertMessage, setPermissionAlertMessage] = useState('');

  const router = useRouter();

  // 新增：消息相关状态

  const [messages, setMessages] = useState<Message[]>([]);

  const [lastMessageId, setLastMessageId] = useState<number>(0);

  const [isFirstLoad, setIsFirstLoad] = useState(true);

  // 使用身份验证 hook

  const { isLoggedIn, username: authUsername, isLoading: authLoading, logout } = useAuth();

  // v2新增：保存最后访问的房间ID到 localStorage

  const saveLastRoom = (roomId: number | null) => {

    if (roomId) {

      localStorage.setItem('lastRoomId', roomId.toString());

    } else {

      localStorage.removeItem('lastRoomId');

    }

  };

  // v2新增读取最后访问的房间ID

  const getLastRoom = (): number | null => {

    const lastRoom = localStorage.getItem('lastRoomId');

    return lastRoom ? parseInt(lastRoom, 10) : null;

  };

  // v2新增更新 selectedRoomId 的设置函数，同时保存到 localStorage

  const handleSelectRoom = (roomId: number | null) => {

    setSelectedRoomId(roomId);

    saveLastRoom(roomId);

  };

  useEffect(() => {

    const saved = localStorage.getItem('nickname');

    if (!saved) {

      alert('请先设置昵称');

      router.push('/setname');

    } else {

      setNickname(saved);

    }

  }, [router]);

  // 检查权限的辅助函数

  const checkPermissionAndAlert = (action: string): boolean => {

    if (!isLoggedIn) {

      setPermissionAlertMessage(`${action}功能需要登录后才能使用，当前为游客模式。`);

      setShowPermissionAlert(true);

      return false;

    }

    return true;

  };

  // 获取房间列表

  const {

    data: roomData,

    error: roomError,

    isLoading: roomIsLoading,

    mutate: refreshRooms,

  } = useSWR<RoomListRes>(

    '/api/room/list',

    (key) => authFetch(key).then((r) => r.json().then((d: any) => d.data)),

    { refreshInterval: 1000 }

  );

  // v2新增：房间列表加载完成后，自动选择上次访问的房间

  useEffect(() => {

    // 只在房间列表首次加载完成且没有选中房间时执行

    if (roomData?.rooms && !selectedRoomId) {

      const lastRoomId = getLastRoom();

      if (lastRoomId) {

        // 检查上次的房间是否仍然存在

        const roomExists = roomData.rooms.some(r => r.roomId === lastRoomId);

        if (roomExists) {

          console.log(`自动进入上次访问的房间: ${lastRoomId}`);

          setSelectedRoomId(lastRoomId);

        } else {

          // 如果上次的房间不存在了，清除记录

          console.log('上次访问的房间已不存在');

          localStorage.removeItem('lastRoomId');

        }

      }

    }

  }, [roomData, selectedRoomId]);

  // v2改进：首次加载获取全部消息

  const {

    data: initialMessages,

    error: initialError,

    isLoading: initialLoading

  } = useSWR<RoomMessageListRes>(

    selectedRoomId && isFirstLoad ? `/api/room/message/list?roomId=${selectedRoomId}` : null,

    (key) => authFetch(key).then((r) => r.json().then((d: any) => d.data)),

    {

      revalidateOnFocus: false,

      revalidateOnReconnect: false

    }

  );

  // v2改进：增量获取新消息

  const { data: newMessages } = useSWR<RoomMessageListRes>(

    selectedRoomId && lastMessageId > 0

      ? `/api/room/message/getUpdate?roomId=${selectedRoomId}&sinceMessageId=${lastMessageId}`

      : null,

    (key) => authFetch(key).then((r) => r.json().then((d: any) => d.data)),

    { refreshInterval: 1000 }

  );

  // v2处理首次加载的消息

  useEffect(() => {

    if (initialMessages?.messages && isFirstLoad) {

      setMessages(initialMessages.messages);

      if (initialMessages.messages.length > 0) {

        const lastMsg = initialMessages.messages[initialMessages.messages.length - 1];

        setLastMessageId(lastMsg.messageId);

      }

      setIsFirstLoad(false);

    }

  }, [initialMessages, isFirstLoad]);

  // v2处理增量更新的消息

  useEffect(() => {

    if (newMessages?.messages && newMessages.messages.length > 0) {

      setMessages(prev => [...prev, ...newMessages.messages]);

      const lastMsg = newMessages.messages[newMessages.messages.length - 1];

      setLastMessageId(lastMsg.messageId);

    }

  }, [newMessages]);

  // v2切换房间时重置状态

  useEffect(() => {

    setMessages([]);

    setLastMessageId(0);

    setIsFirstLoad(true);

  }, [selectedRoomId]);

  // 创建房间

  const { trigger: addRoomTrigger, isMutating: isAddingRoom } = useSWRMutation<

    RoomAddRes,

    Error,

    string,

    RoomAddArgs

  >('/api/room/add', (key, { arg }) =>

    authFetch(key, {

      method: 'POST',

      body: JSON.stringify(arg),

    }).then((r) => r.json().then((d: any) => d.data))

  );

  // 删除房间

  const { trigger: deleteRoomTrigger } = useSWRMutation(

    '/api/room/delete',

    async (key, { arg }: { arg: { roomId: number } }) => {

      const response = await authFetch(key, {

        method: 'POST',

        body: JSON.stringify({ roomId: arg.roomId }),

      });

      return response;

    }

  );

  // 发送消息

  const { trigger: sendMessageTrigger, isMutating: isSendingMessage } =

    useSWRMutation<null, Error, string, MessageAddArgs>(

      '/api/message/add',

      (key, { arg }) =>

        authFetch(key, {

          method: 'POST',

          body: JSON.stringify(arg),

        }).then((r) => r.json().then((d: any) => d.data))

    );

  // 处理显示添加房间模态框

  const handleShowAddModal = () => {

    if (!checkPermissionAndAlert('创建房间')) return;

    setShowAddModal(true);

  };

  // 处理创建房间

  const handleAddRoom = async () => {

    if (!newRoomName.trim()) return alert('请输入房间名称');

    if (!checkPermissionAndAlert('创建房间')) {

      setShowAddModal(false);

      return;

    }

    try {

      const res = await addRoomTrigger({ user: nickname, roomName: newRoomName.trim() });

      refreshRooms();

      if (res?.roomId) {

        handleSelectRoom(res.roomId); //使用新的设置函数

      }

      setShowAddModal(false);

      setNewRoomName('');

    } catch (error) {

      console.error('创建房间失败:', error);

      alert('创建房间失败');

    }

  };

  // 处理删除房间

  const handleDeleteRoom = async (roomId: number) => {

    if (!isLoggedIn) {

      alert('请先登录后再删除房间');

      return;

    }

    const token = localStorage.getItem('token');

    if (!token) {

      alert('登录状态已过期，请重新登录');

      logout();

      return;

    }

    if (!confirm('确定删除该房间？此操作不可撤销！')) return;

    try {

      await deleteRoomTrigger({ roomId });

      if (selectedRoomId === roomId) {

        handleSelectRoom(null); // 使用新的设置函数

      }

      refreshRooms();

      alert('房间删除成功');

    } catch (error) {

      console.error('删除房间失败:', error);

      if (error instanceof Error) {

        if (error.message.includes('认证') || error.message.includes('token')) {

          alert('认证失败，请重新登录');

          logout();

        } else {

          alert(`删除房间失败: ${error.message}`);

        }

      } else {

        alert('删除房间失败，请稍后重试');

      }

    }

  };

  //处理发送消息

  const handleSendMessage = async () => {

    if (!messageInput.trim() || !selectedRoomId) return;

    try {

      await sendMessageTrigger({

        roomId: selectedRoomId,

        content: messageInput.trim(),

        sender: nickname,

      });

      setMessageInput('');

      // 发送成功后，增量更新会自动获取新消息

      // 不需要手动刷新

    } catch {

      alert('发送消息失败');

    }

  };

  const handleKeyPress = (e: React.KeyboardEvent<HTMLInputElement>) => {

    if (e.key === 'Enter' && !e.shiftKey) {

      e.preventDefault();

      handleSendMessage();

    }

  };

  if (authLoading) {

    return <div className={styles.loading}>正在检查登录状态...</div>;

  }

  //计算加载和错误状态

  const messageIsLoading = initialLoading && isFirstLoad;

  const messageError = initialError;

  return (

    <main className={styles.chatContainer}>

      {/\* 身份验证状态显示 \*/}

      <AuthStatus

        isLoggedIn={isLoggedIn}

        username={authUsername}

        onLogout={logout}

      />

      {/\* 左侧房间列表 - 修改点击事件 \*/}

      <div className={styles.sidebar}>

        <div className={styles.sidebarHeader}>

          <h2>聊天室</h2>

          <button

            className={styles.addButton}

            onClick={handleShowAddModal}

            title={isLoggedIn ? "创建房间" : "需要登录才能创建房间"}

            disabled={!isLoggedIn}

          >

            +

          </button>

        </div>

        <div className={styles.roomList}>

          {roomIsLoading && <div className={styles.loading}>加载中...</div>}

          {roomError && <div className={styles.error}>加载房间失败</div>}

          {roomData?.rooms?.map((r) => (

            <RoomEntry

              key={r.roomId}

              room={r}

              onClick={() => handleSelectRoom(r.roomId)} // 使用新的设置函数

              isActive={selectedRoomId === r.roomId}

              onDelete={() => handleDeleteRoom(r.roomId)}

              canDelete={isLoggedIn}

            />

          ))}

        </div>

      </div>

      {/\* 右侧聊天窗口 \*/}

      <div className={styles.chatArea}>

        {selectedRoomId ? (

          <div className={styles.chatContent}>

            <div className={styles.chatHeader}>

              <h3>

                {roomData?.rooms.find((r) => r.roomId === selectedRoomId)?.roomName ||

                  `房间 ${selectedRoomId}`}

              </h3>

            </div>

            <div className={styles.messagesContainer}>

              {messageIsLoading && <div className={styles.loading}>加载消息中...</div>}

              {messageError && <div className={styles.error}>加载消息失败</div>}

              {messages.map((m) => (

                <MessageItem key={m.messageId} message={m} currentUser={nickname} />

              ))}

            </div>

            <div className={styles.messageInput}>

              <input

                type="text"

                placeholder="输入消息..."

                value={messageInput}

                onChange={(e) => setMessageInput(e.target.value)}

                onKeyPress={handleKeyPress}

                disabled={isSendingMessage}

                className={styles.textInput}

              />

              <button

                onClick={handleSendMessage}

                disabled={isSendingMessage || !messageInput.trim()}

                className={styles.sendButton}

              >

                {isSendingMessage ? '发送中...' : '发送'}

              </button>

            </div>

          </div>

        ) : (

          <div className={styles.noRoom}>请选择一个聊天室开始聊天</div>

        )}

      </div>

      {/\* 创建房间弹窗 \*/}

      {showAddModal && (

        <div className={styles.modal} onClick={() => setShowAddModal(false)}>

          <div className={styles.modalContent} onClick={(e) => e.stopPropagation()}>

            <h3>创建新房间</h3>

            <input

              type="text"

              placeholder="房间名称"

              value={newRoomName}

              onChange={(e) => setNewRoomName(e.target.value)}

              className={styles.input}

            />

            <div className={styles.modalButtons}>

              <button onClick={handleAddRoom} disabled={isAddingRoom} className={styles.confirmButton}>

                {isAddingRoom ? '创建中...' : '创建'}

              </button>

              <button onClick={() => setShowAddModal(false)} className={styles.cancelButton}>

                取消

              </button>

            </div>

          </div>

        </div>

      )}

      {/\* 权限提示弹窗 \*/}

      {showPermissionAlert && (

        <div className={styles.modal}>

          <div className={styles.modalContent}>

            <PermissionAlert

              message={permissionAlertMessage}

              onClose={() => setShowPermissionAlert(false)}

            />

          </div>

        </div>

      )}

    </main>

  );

}

**src/app/chat/page.module.css：**

.chatContainer {

  display: flex;

  height: 100vh;

  max-width: 1200px;

  margin: 0 auto;

  background-color: #f5f5f5;

}

.sidebar {

  width: 300px;

  background-color: white;

  border-right: 1px solid #e1e1e1;

  display: flex;

  flex-direction: column;

}

.sidebarHeader {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 1rem;

  border-bottom: 1px solid #e1e1e1;

}

.sidebarHeader h2 {

  margin: 0;

  font-size: 1.2rem;

  color: #333;

}

.addButton {

  background: #0070f3;

  color: white;

  border: none;

  border-radius: 50%;

  width: 30px;

  height: 30px;

  font-size: 1.2rem;

  cursor: pointer;

  transition: background 0.3s ease;

}

.addButton:hover {

  background: #005bb5;

}

.addButton:disabled {

  background: #ccc;

  cursor: not-allowed;

}

.roomList {

  flex: 1;

  overflow-y: auto;

  padding: 0.5rem 0;

}

.roomItem {

  padding: 1rem;

  border-bottom: 1px solid #f0f0f0;

  cursor: pointer;

  transition: background 0.2s ease;

}

.roomItem:hover {

  background-color: #f8f9fa;

}

.roomItem.active {

  background-color: #e3f2fd;

  border-left: 3px solid #0070f3;

}

.roomName {

  margin: 0 0 0.5rem 0;

  font-size: 1rem;

  font-weight: 500;

  color: #333;

}

.lastMessage {

  margin: 0;

  font-size: 0.85rem;

  color: #666;

  white-space: nowrap;

  overflow: hidden;

  text-overflow: ellipsis;

}

.chatArea {

  flex: 1;

  display: flex;

  flex-direction: column;

  background-color: white;

}

.chatContent {

  flex: 1;

  display: flex;

  flex-direction: column;

  height: 100vh;

}

.chatHeader {

  padding: 1rem;

  border-bottom: 1px solid #e1e1e1;

  background-color: white;

}

.chatHeader h3 {

  margin: 0;

  font-size: 1.2rem;

  color: #333;

}

.messagesContainer {

  flex: 1;

  overflow-y: auto;

  padding: 1rem;

  background-color: #f8f9fa;

}

.messageItem {

  margin-bottom: 1rem;

  padding: 0.75rem;

  border-radius: 8px;

  max-width: 70%;

}

.ownMessage {

  background-color: #0070f3;

  color: white;

  margin-left: auto;

  text-align: right;

}

.otherMessage {

  background-color: white;

  color: #333;

  border: 1px solid #e1e1e1;

}

.messageHeader {

  display: flex;

  justify-content: space-between;

  margin-bottom: 0.25rem;

  font-size: 0.8rem;

}

.ownMessage .messageHeader {

  color: rgba(255, 255, 255, 0.8);

}

.otherMessage .messageHeader {

  color: #666;

}

.sender {

  font-weight: 500;

}

.time {

  font-size: 0.75rem;

}

.messageContent {

  word-wrap: break-word;

  line-height: 1.4;

}

.messageInput {

  display: flex;

  padding: 1rem;

  border-top: 1px solid #e1e1e1;

  background-color: white;

  gap: 0.5rem;

}

.textInput {

  flex: 1;

  padding: 0.75rem;

  border: 1px solid #ddd;

  border-radius: 20px;

  font-size: 1rem;

  outline: none;

}

.textInput:focus {

  border-color: #0070f3;

}

.textInput:disabled {

  background-color: #f5f5f5;

  cursor: not-allowed;

}

.sendButton {

  padding: 0.75rem 1.5rem;

  background: #0070f3;

  color: white;

  border: none;

  border-radius: 20px;

  font-size: 1rem;

  cursor: pointer;

  transition: background 0.3s ease;

  white-space: nowrap;

}

.sendButton:hover {

  background: #005bb5;

}

.sendButton:disabled {

  background: #ccc;

  cursor: not-allowed;

}

.noRoom {

  display: flex;

  align-items: center;

  justify-content: center;

  flex: 1;

  color: #666;

  font-size: 1.1rem;

}

.loading, .empty {

  padding: 2rem;

  text-align: center;

  color: #666;

}

.error {

  padding: 2rem;

  text-align: center;

  color: #d32f2f;

  background-color: #ffebee;

  margin: 1rem;

  border-radius: 4px;

}

/\* 模态框样式 \*/

.modal {

  position: fixed;

  top: 0;

  left: 0;

  right: 0;

  bottom: 0;

  background-color: rgba(0, 0, 0, 0.5);

  display: flex;

  align-items: center;

  justify-content: center;

  z-index: 1000;

}

.modalContent {

  background: white;

  padding: 2rem;

  border-radius: 8px;

  width: 90%;

  max-width: 400px;

  box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);

}

.modalContent h3 {

  margin: 0 0 1rem 0;

  color: #333;

  text-align: center;

}

.input {

  width: 100%;

  padding: 0.75rem;

  border: 1px solid #ddd;

  border-radius: 4px;

  font-size: 1rem;

  margin-bottom: 1.5rem;

  box-sizing: border-box;

}

.input:focus {

  outline: none;

  border-color: #0070f3;

}

.input:disabled {

  background-color: #f5f5f5;

  cursor: not-allowed;

}

.modalButtons {

  display: flex;

  gap: 1rem;

  justify-content: center;

}

.confirmButton, .cancelButton {

  padding: 0.75rem 1.5rem;

  border: none;

  border-radius: 4px;

  font-size: 1rem;

  cursor: pointer;

  transition: background 0.3s ease;

  min-width: 80px;

}

.confirmButton {

  background: #0070f3;

  color: white;

}

.confirmButton:hover {

  background: #005bb5;

}

.confirmButton:disabled {

  background: #ccc;

  cursor: not-allowed;

}

.cancelButton {

  background: #f5f5f5;

  color: #333;

}

.cancelButton:hover {

  background: #e0e0e0;

}

.cancelButton:disabled {

  background: #f9f9f9;

  cursor: not-allowed;

}

/\* 响应式设计 \*/

@media (max-width: 768px) {

  .chatContainer {

    flex-direction: column;

  }

  .sidebar {

    width: 100%;

    height: 40vh;

  }

  .chatArea {

    height: 60vh;

  }

  .messageItem {

    max-width: 85%;

  }

  .messageInput {

    padding: 0.75rem;

  }

  .textInput {

    font-size: 16px; /\* 防止iOS缩放 \*/

  }

}

.roomItem {

  position: relative; /\* 为了定位删除按钮 \*/

  padding: 1rem;

  border-bottom: 1px solid #f0f0f0;

  cursor: pointer;

  transition: background 0.2s ease;

}

.roomItem:hover {

  background-color: #f8f9fa;

}

.roomItem:hover .deleteBtn {

  opacity: 1; /\* 悬停时显示删除按钮 \*/

}

.roomItem.active {

  background-color: #e3f2fd;

  border-left: 3px solid #0070f3;

}

.deleteBtn {

  position: absolute;

  top: 50%;

  right: 10px;

  transform: translateY(-50%);

  background: #ff4444;

  color: white;

  border: none;

  border-radius: 50%;

  width: 20px;

  height: 20px;

  font-size: 12px;

  cursor: pointer;

  opacity: 0; /\* 默认隐藏 \*/

  transition: opacity 0.2s ease, background 0.2s ease;

  display: flex;

  align-items: center;

  justify-content: center;

  line-height: 1;

}

.deleteBtn:hover {

  background: #cc0000;

}

.deleteBtn:active {

  transform: translateY(-50%) scale(0.95);

}

/\* 确保房间名称不会与删除按钮重叠 \*/

.roomName {

  margin: 0 0 0.5rem 0;

  font-size: 1rem;

  font-weight: 500;

  color: #333;

  padding-right: 30px; /\* 为删除按钮留出空间 \*/

}

.lastMessage {

  margin: 0;

  font-size: 0.85rem;

  color: #666;

  white-space: nowrap;

  overflow: hidden;

  text-overflow: ellipsis;

  padding-right: 30px; /\* 为删除按钮留出空间 \*/

}

**chatroom/src/app/greet/page.tsx：**

'use client';

import { useEffect, useState } from 'react';

import { useRouter } from 'next/navigation';

import './greet.css';

export default function ChatPage() {

  const [nickname, setNickname] = useState('');

  const router = useRouter();

  useEffect(() => {

    const savedNickname = localStorage.getItem('nickname');

    if (!savedNickname) {

      alert('请先设置昵称');

      router.push('/nickname');

    } else {

      setNickname(savedNickname);

    }

  }, []);

    useEffect(() => {

    setTimeout(() => {

      // 5秒后跳转到目标页面

      router.push('/chat')

    }, 3000); // 5000毫秒（5秒）后执行

  }, []); // 空依赖数组表示这个effect只会在组件挂载时执行一次

  return (

    <div>

        <h1 className='{styles.title}'>欢迎，{nickname}！</h1>

            <p>这个页面将在3秒后自动跳转聊天页面</p>

    </div>

  );

}

**chatroom/src/app/greet/greet.css：**

.div{

    color: #c3aae2;

    text-align: center;

    padding: 20px;

}

.title{

    font-size: 2rem;

    color: #25044e;

    margin-bottom: 20px;

}

**chatroom/src/app/login/page.tsx：**

'use client';

import { useState } from 'react';

import { useRouter } from 'next/navigation';

export default function LoginPage() {

  const [username, setU] = useState('');

  const [password, setP] = useState('');

  const router = useRouter();

  const login = async () => {

    const res = await fetch('/api/auth/login', {

      method: 'POST',

      headers: { 'Content-Type': 'application/json' },

      body: JSON.stringify({ username, password }),

    });

    const json = await res.json();

    if (json.code !== 0) return alert(json.message);

    localStorage.setItem('token', json.data.accessToken);

    localStorage.setItem('nickname', username);

    router.push('/setname'); // 或直接 /chat

  };

  return (

    <main style={{ padding: '2rem', textAlign: 'center' }}>

      <h1>登录</h1>

      <input placeholder="用户名" value={username} onChange={e => setU(e.target.value)} /><br/>

      <input placeholder="密码" type="password" value={password} onChange={e => setP(e.target.value)} /><br/>

      <button onClick={login}>登录</button>

      <p>没账号？<a href="/signup">去注册</a></p>

    </main>

  );

}

**chatroom/src/app/setname/page.tsx：**

'use client';

import { useState } from 'react';

import { useRouter } from 'next/navigation';

export default function NicknamePage() {

  const [nickname, setNickname] = useState('');

  const router = useRouter();

  const handleEnterChat = () => {

    if (!nickname.trim()) {

      alert('请输入昵称');

      return;

    }

    // 将昵称保存在 localStorage 中（暂时作为模拟）

    localStorage.setItem('nickname', nickname);

    // 跳转到聊天室页面

    router.push('/greet');

  };

  return (

    <main style={{ padding: '2rem', textAlign: 'center' }}>

      <h1>设置你的昵称</h1>

      <input

        type="text"

        placeholder="请输入昵称"

        value={nickname}

        onChange={(e) => setNickname(e.target.value)}

        style={{

          padding: '0.5rem',

          fontSize: '1rem',

          width: '200px',

          margin: '1rem 0',

        }}

      />

      <br />

      <button

        onClick={handleEnterChat}

        style={{

          padding: '0.5rem 1rem',

          fontSize: '1rem',

          cursor: 'pointer',

        }}

      >

        进入聊天室

      </button>

    </main>

  );

}

**chatroom/src/app/signup/page.tsx：**

'use client';

import { useState } from 'react';

import { useRouter } from 'next/navigation';

export default function SignupPage() {

  const [username, setU] = useState('');

  const [password, setP] = useState('');

  const [confirmPwd, setC] = useState('');

  const router = useRouter();

  const signup = async () => {

    if (!username.trim() || !password.trim()) return alert('用户名和密码不能为空');

    if (password !== confirmPwd) return alert('两次密码不一致');

    const res = await fetch('/api/auth/signup', {

      method: 'POST',

      headers: { 'Content-Type': 'application/json' },

      body: JSON.stringify({ username, password }),

    });

    const json = await res.json();

    if (json.code !== 0) return alert(json.message);

    alert('注册成功！请登录');

    router.push('/login');

  };

  return (

    <main style={{ padding: '2rem', textAlign: 'center' }}>

      <h1>注册账号</h1>

      <input placeholder="用户名" value={username} onChange={e => setU(e.target.value)} /><br />

      <input placeholder="密码" type="password" value={password} onChange={e => setP(e.target.value)} /><br />

      <input placeholder="确认密码" type="password" value={confirmPwd} onChange={e => setC(e.target.value)} /><br />

      <button onClick={signup}>注册</button>

      <p>已有账号？<a href="/login">去登录</a></p>

    </main>

  );

}

**chatroom/src/app/page.tsx：**

// 导入Next.js的Link组件（客户端导航，比<a>标签性能更好）

import Link from 'next/link';

// 导入CSS模块

import styles from './page.module.css';

// 默认导出的首页组件

export default function Home() {

  return (

    <main className={styles.container}>

      {/\* 标题部分 \*/}

      <h1 className={styles.title}>欢迎来到JYCHATROOM</h1>

      {/\* 按钮容器 \*/}

      <div className={styles.buttonGroup}>

        {/\* 跳转到设置页的按钮 \*/}

        <Link href="/setname" className={styles.button}>

          设置昵称

        </Link>

        {/\* 跳转到聊天室的按钮 \*/}

        <Link href="/greet" className={styles.button}>

          进入聊天室

        </Link>

        <Link href="/login" className={styles.button}>

          登录

        </Link>

      </div>

    </main>

  );

}