React 聊天应用开发完整流程

项目整体架构

```
chat-app/
├── src/
│   ├── components/          # 可复用组件
│   │   ├── ChatMessage.jsx
│   │   ├── MessageInput.jsx
│   │   ├── UserList.jsx
│   │   └── ProtectedRoute.jsx
│   ├── pages/               # 页面组件
│   │   ├── Login.jsx
│   │   ├── Chat.jsx
│   │   └── Profile.jsx
│   ├── store/               # Redux 相关
│   │   ├── index.js         # store 配置
│   │   ├── slices/          # Redux Toolkit slices
│   │   │   ├── authSlice.js
│   │   │   ├── chatSlice.js
│   │   │   └── userSlice.js
│   │   └── middleware/      # 中间件
│   │       └── api.js
│   ├── services/            # API 服务
│   │   ├── auth.js
│   │   ├── chat.js
│   │   └── api.js
│   ├── hooks/               # 自定义 hooks
│   │   ├── useAuth.js
│   │   └── useSocket.js
│   ├── utils/               # 工具函数
│   │   ├── constants.js
│   │   └── helpers.js
│   ├── App.jsx              # 根组件
│   └── main.jsx             # 入口文件
├── package.json
└── vite.config.js
```

第一步：项目初始化

1.1 创建 Vite 项目

bash

```
npm create vite@latest chat-app -- --template react
cd chat-app
npm install
```

1.2 安装必要依赖

bash

```bash
# 核心依赖
npm install @reduxjs/toolkit react-redux react-router-dom axios

# 可选依赖 (根据需要)
npm install socket.io-client  # 如果使用Socket.IO
npm install classnames       # 用于动态类名
```

知识点标记：

- `@reduxjs/toolkit`: Redux的现代工具包，简化Redux使用
- `react-redux`: React和Redux的连接库
- `react-router-dom`: React路由管理
- `axios`: HTTP请求库

第二步：配置 Vite

2.1 配置 vite.config.js

javascript

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { resolve } from 'path'

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': resolve(__dirname, 'src'),
    },
  },
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true,
      },
    },
  },
})
```

知识点标记：

- `alias`: 路径别名配置，可以用 @/ 代替 src/
- `proxy`: 开发环境代理配置，解决跨域问题

第三步：设置 Redux Store

3.1 创建 store/index.js

javascript

```javascript
import { configureStore } from '@reduxjs/toolkit'
import authSlice from './slices/authSlice'
import chatSlice from './slices/chatSlice'
import userSlice from './slices/userSlice'

export const store = configureStore({
  reducer: {
    auth: authSlice,
    chat: chatSlice,
    user: userSlice,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: ['persist/PERSIST'],
      },
    }),
})

// 如果需要在其他文件中使用store类型，可以这样导出
// export const selectAuth = (state) => state.auth
// export const selectChat = (state) => state.chat
```

知识点标记：

- `configureStore`: Redux Toolkit的store配置函数
- `reducer`: 合并多个slice reducer
- `middleware`: 中间件配置
- `RootState`: TypeScript类型定义

3.2 创建 store/slices/authSlice.js

javascript

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
import * as authAPI from '../../services/auth'

// 异步登录action
export const loginUser = createAsyncThunk(
  'auth/login',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const response = await authAPI.login(email, password)
      return response.data
    } catch (error) {
      return rejectWithValue(error.response.data)
    }
  }
)

// 异步登出action
export const logoutUser = createAsyncThunk(
  'auth/logout',
  async (_, { rejectWithValue }) => {
    try {
      await authAPI.logout()
    } catch (error) {
      return rejectWithValue(error.response.data)
    }
  }
)

const authSlice = createSlice({
  name: 'auth',
  initialState: {
    user: null,
    token: localStorage.getItem('token'),
    isLoading: false,
    error: null,
    isAuthenticated: !!localStorage.getItem('token'),
  },
  reducers: {
    // 同步actions
    clearError: (state) => {
      state.error = null
    },
    setCredentials: (state, action) => {
      state.user = action.payload.user
      state.token = action.payload.token
      state.isAuthenticated = true
    },
  },
  extraReducers: (builder) => {
    builder
      // 登录相关
      .addCase(loginUser.pending, (state) => {
        state.isLoading = true
        state.error = null
      })
      .addCase(loginUser.fulfilled, (state, action) => {
        state.isLoading = false
        state.user = action.payload.user
        state.token = action.payload.token
        state.isAuthenticated = true
        localStorage.setItem('token', action.payload.token)
      })
      .addCase(loginUser.rejected, (state, action) => {
        state.isLoading = false
        state.error = action.payload.message
      })
      // 登出相关
      .addCase(logoutUser.fulfilled, (state) => {
        state.user = null
        state.token = null
        state.isAuthenticated = false
        localStorage.removeItem('token')
      })
  },
})

export const { clearError, setCredentials } = authSlice.actions
export default authSlice.reducer
```

知识点标记：

- `createSlice`: 创建slice，包含reducer和actions
- `createAsyncThunk`: 创建异步action
- `extraReducers`: 处理异步action的状态变化
- `builder pattern`: 现代Redux Toolkit的reducer写法

3.3 创建 store/slices/chatSlice.js

javascript

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
import * as chatAPI from '../../services/chat'

export const fetchMessages = createAsyncThunk(
  'chat/fetchMessages',
  async (roomId, { rejectWithValue }) => {
    try {
      const response = await chatAPI.getMessages(roomId)
      return response.data
    } catch (error) {
      return rejectWithValue(error.response.data)
    }
  }
)

export const sendMessage = createAsyncThunk(
  'chat/sendMessage',
  async ({ roomId, message }, { rejectWithValue }) => {
    try {
      const response = await chatAPI.sendMessage(roomId, message)
      return response.data
    } catch (error) {
      return rejectWithValue(error.response.data)
    }
  }
)

const chatSlice = createSlice({
  name: 'chat',
  initialState: {
    messages: [],
    currentRoom: null,
    isLoading: false,
    error: null,
    onlineUsers: [],
  },
  reducers: {
    setCurrentRoom: (state, action) => {
      state.currentRoom = action.payload
    },
    addMessage: (state, action) => {
      state.messages.push(action.payload)
    },
    updateOnlineUsers: (state, action) => {
      state.onlineUsers = action.payload
    },
    clearMessages: (state) => {
      state.messages = []
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchMessages.pending, (state) => {
        state.isLoading = true
      })
      .addCase(fetchMessages.fulfilled, (state, action) => {
        state.isLoading = false
        state.messages = action.payload
      })
      .addCase(fetchMessages.rejected, (state, action) => {
        state.isLoading = false
        state.error = action.payload.message
      })
      .addCase(sendMessage.fulfilled, (state, action) => {
        state.messages.push(action.payload)
      })
  },
})

export const { setCurrentRoom, addMessage, updateOnlineUsers, clearMessages } = chatSlice.actions
export default chatSlice.reducer
```

第四步：配置路由

4.1 创建 App.jsx

javascript

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
import { Provider } from 'react-redux'
import { store } from './store'
import Login from './pages/Login'
import Chat from './pages/Chat'
import Profile from './pages/Profile'
import ProtectedRoute from './components/ProtectedRoute'
import './App.css'

function App() {
  return (
    <Provider store={store}>
      <BrowserRouter>
        <div className="App">
          <Routes>
            <Route path="/login" element={<Login />} />
            <Route
              path="/chat"
              element={
                <ProtectedRoute>
                  <Chat />
                </ProtectedRoute>
              }
            />
            <Route
              path="/profile"
              element={
                <ProtectedRoute>
                  <Profile />
                </ProtectedRoute>
              }
            />
            <Route path="/" element={<Navigate to="/chat" replace />} />
          </Routes>
        </div>
      </BrowserRouter>
    </Provider>
  )
}

export default App
```

知识点标记：

- `BrowserRouter`: 使用HTML5 history API的路由器

- `Routes/Route`: 路由配置
- `Navigate`: 程序式导航组件
- `Provider`: Redux的上下文提供者
- `ProtectedRoute`: 路由保护组件

4.2 创建 components/ProtectedRoute.jsx

javascript

```javascript
import { useSelector } from 'react-redux'
import { Navigate } from 'react-router-dom'

const ProtectedRoute = ({ children }) => {
  const isAuthenticated = useSelector(state => state.auth.isAuthenticated)

  return isAuthenticated ? children : <Navigate to="/login" replace />
}

export default ProtectedRoute
```

知识点标记：

- `useSelector`: 从Redux store中选择状态
- 路由守卫：检查认证状态决定是否允许访问

第五步：创建API服务

5.1 创建 services/api.js

javascript

```
import axios from 'axios'
import { store } from '../store'

// 创建axios实例
const api = axios.create({
  baseURL: '/api',
  timeout: 10000,
})

// 请求拦截器
api.interceptors.request.use(
  (config) => {
    const token = store.getState().auth.token
    if (token) {
      config.headers.Authorization = `Bearer ${token}`
    }
    return config
  },
  (error) => {
    return Promise.reject(error)
  }
)

// 响应拦截器
api.interceptors.response.use(
  (response) => {
    return response
  },
  (error) => {
    if (error.response?.status === 401) {
      // Token过期，清除认证信息
      store.dispatch({ type: 'auth/logout' })
    }
    return Promise.reject(error)
  }
)

export default api
```

知识点标记：

- `axios.create()`: 创建axios实例
- `interceptors`: 请求/响应拦截器
- `timeout`: 请求超时设置
- `Authorization`: JWT token认证

5.2 创建 services/auth.js

javascript

```javascript
import api from './api'

export const login = async (email, password) => {
  return await api.post('/auth/login', { email, password })
}

export const register = async (userData) => {
  return await api.post('/auth/register', userData)
}

export const logout = async () => {
  return await api.post('/auth/logout')
}

export const refreshToken = async () => {
  return await api.post('/auth/refresh')
}
```

5.3 创建 services/chat.js

javascript

```javascript
import api from './api'

export const getMessages = async (roomId) => {
  return await api.get(`/chat/messages/${roomId}`)
}

export const sendMessage = async (roomId, message) => {
  return await api.post(`/chat/messages/${roomId}`, { message })
}

export const getRooms = async () => {
  return await api.get('/chat/rooms')
}

export const createRoom = async (roomData) => {
  return await api.post('/chat/rooms', roomData)
}
```

第六步：创建页面组件

6.1 创建 pages/Login.jsx

javascript

```
import { useState, useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { useNavigate } from 'react-router-dom'
import { loginUser, clearError } from '../store/slices/authSlice'

const Login = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
  })

  const dispatch = useDispatch()
  const navigate = useNavigate()
  const { isLoading, error, isAuthenticated } = useSelector(state => state.auth)

  useEffect(() => {
    if (isAuthenticated) {
      navigate('/chat')
    }
  }, [isAuthenticated, navigate])

  useEffect(() => {
    return () => {
      dispatch(clearError())
    }
  }, [dispatch])

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    })
  }

  const handleSubmit = async (e) => {
    e.preventDefault()
    dispatch(loginUser(formData))
  }

  return (
    <div className="login-container">
      <form onSubmit={handleSubmit} className="login-form">
        <h2>登录</h2>

        {error && <div className="error-message">{error}</div>}
        <div className="form-group">
          <input
            type="email"
            name="email"
            placeholder="邮箱"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>

        <div className="form-group">
          <input
            type="password"
            name="password"
            placeholder="密码"
            value={formData.password}
            onChange={handleChange}
            required
          />
        </div>

        <button type="submit" disabled={isLoading}>
          {isLoading ? '登录中...' : '登录'}
        </button>
      </form>
    </div>
  )
}

export default Login
```

知识点标记：

- `useDispatch`: 获取dispatch函数
- `useSelector`: 选择Redux状态
- `useNavigate`: 程序式导航
- `useEffect`: 副作用处理
- 表单受控组件

6.2 创建 pages/Chat.jsx

javascript

```
import { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { fetchMessages, setCurrentRoom } from '../store/slices/chatSlice'
import ChatMessage from '../components/ChatMessage'
import MessageInput from '../components/MessageInput'
import UserList from '../components/UserList'
import useSocket from '../hooks/useSocket'

const Chat = () => {
  const dispatch = useDispatch()
  const { messages, currentRoom, isLoading, onlineUsers } = useSelector(state => state.chat)
  const { user } = useSelector(state => state.auth)

  // 使用WebSocket hook
  useSocket()

  useEffect(() => {
    // 设置默认房间
    if (!currentRoom) {
      dispatch(setCurrentRoom('general'))
    }
  }, [currentRoom, dispatch])

  useEffect(() => {
    // 获取消息
    if (currentRoom) {
      dispatch(fetchMessages(currentRoom))
    }
  }, [currentRoom, dispatch])

  return (
    <div className="chat-container">
      <div className="chat-sidebar">
        <UserList users={onlineUsers} />
      </div>

      <div className="chat-main">
        <div className="chat-header">
          <h3>#{currentRoom}</h3>
          <span>{onlineUsers.length} 在线</span>
        </div>

        <div className="chat-messages">
          {isLoading ? (
            <div>加載中...</div>
          ) : (
          messages.map(message => (
            <ChatMessage
              key={message.id}
              message={message}
              isOwn={message.userId === user?.id}
            />
          ))
        )}
      </div>

      <MessageInput />
    </div>
  </div>
 )
}

export default Chat
```

知识点标记：

- 组件组合：将大组件拆分为小组件
- 条件渲染：根据状态显示不同内容
- 列表渲染：map函数渲染消息列表
- 自定义hook：封装WebSocket逻辑

第七步：创建可复用组件

7.1 创建 components/ChatMessage.jsx

javascript

```
import { formatTime } from '../utils/helpers'

const ChatMessage = ({ message, isOwn }) => {
  return (
    <div className={`message ${isOwn ? 'message-own' : 'message-other'}`}>
      <div className="message-avatar">
        {message.user.avatar ? (
          <img src={message.user.avatar} alt={message.user.name} />
        ) : (
          <div className="avatar-placeholder">
            {message.user.name.charAt(0).toUpperCase()}
          </div>
        )}
      </div>

      <div className="message-content">
        <div className="message-header">
          <span className="message-author">{message.user.name}</span>
          <span className="message-time">{formatTime(message.createdAt)}</span>
        </div>
        <div className="message-text">{message.text}</div>
      </div>
    </div>
  )
}

export default ChatMessage
```

7.2 创建 components/MessageInput.jsx

javascript

```
import { useState } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { sendMessage } from '../store/slices/chatSlice'

const MessageInput = () => {
  const [message, setMessage] = useState('')
  const dispatch = useDispatch()
  const { currentRoom } = useSelector(state => state.chat)

  const handleSubmit = (e) => {
    e.preventDefault()
    if (message.trim() && currentRoom) {
      dispatch(sendMessage({
        roomId: currentRoom,
        message: message.trim(),
      }))
      setMessage('')
    }
  }

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault()
      handleSubmit(e)
    }
  }

  return (
    <form onSubmit={handleSubmit} className="message-input-container">
      <input
        type="text"
        value={message}
        onChange={(e) => setMessage(e.target.value)}
        onKeyPress={handleKeyPress}
        placeholder="输入消息..."
        className="message-input"
      />
      <button type="submit" disabled={!message.trim()}>
        发送
      </button>
    </form>
  )
}

export default MessageInput
```

7.3 创建 components/UserList.jsx

javascript

```javascript
const UserList = ({ users }) => {
  return (
    <div className="user-list">
      <h3>在线用户</h3>
      <div className="users">
        {users.map(user => (
          <div key={user.id} className="user-item">
            <div className="user-avatar">
              {user.avatar ? (
                <img src={user.avatar} alt={user.name} />
              ) : (
                <div className="avatar-placeholder">
                  {user.name.charAt(0).toUpperCase()}
                </div>
              )}
            </div>
            <span className="user-name">{user.name}</span>
            <span className="user-status online"></span>
          </div>
        ))}
      </div>
    </div>
  )
}

export default UserList
```

知识点标记：

- 组件props传递
- 条件渲染
- 数组map渲染
- 默认头像处理

8.1 创建 hooks/useSocket.js

javascript

```
import { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { addMessage, updateOnlineUsers } from '../store/slices/chatSlice'

const useSocket = () => {
  const dispatch = useDispatch()
  const { token } = useSelector(state => state.auth)
  const { currentRoom } = useSelector(state => state.chat)

  useEffect(() => {
    if (!token) return

    // 模拟WebSocket连接 - 在实际项目中使用 socket.io-client
    const connectWebSocket = () => {
      const ws = new WebSocket(`ws://localhost:5000?token=${token}`)

      ws.onopen = () => {
        console.log('Connected to server')
        if (currentRoom) {
          ws.send(JSON.stringify({ type: 'join_room', room: currentRoom }))
        }
      }

      ws.onmessage = (event) => {
        const data = JSON.parse(event.data)
        switch (data.type) {
          case 'new_message':
            dispatch(addMessage(data.message))
            break
          case 'users_update':
            dispatch(updateOnlineUsers(data.users))
            break
          default:
            break
        }
      }

      ws.onclose = () => {
        console.log('Disconnected from server')
      }

      ws.onerror = (error) => {
        console.error('WebSocket error:', error)
      }

      return ws
```

```javascript
    }

    const socket = connectWebSocket()

    return () => {
      socket.close()
    }
  }, [token, currentRoom, dispatch])
}

export default useSocket
```

知识点标记：

- 自定义hook：封装复杂逻辑
- WebSocket连接管理
- 事件监听和清理
- 依赖数组管理

8.2 创建 hooks/useAuth.js

javascript

```javascript
import { useSelector } from 'react-redux'

const useAuth = () => {
  const { user, isAuthenticated, isLoading } = useSelector(state => state.auth)

  return {
    user,
    isAuthenticated,
    isLoading,
    isAdmin: user?.role === 'admin',
    userId: user?.id,
  }
}

export default useAuth
```

第九步：工具函数

9.1 创建 utils/helpers.js

javascript

```
export const formatTime = (timestamp) => {
  const date = new Date(timestamp)
  const now = new Date()
  const diff = now - date

  if (diff < 60000) { // 1分钟内
    return '刚刚'
  } else if (diff < 3600000) { // 1小时内
    return `${Math.floor(diff / 60000)}分钟前`
  } else if (diff < 86400000) { // 24小时内
    return date.toLocaleTimeString('zh-CN', { hour: '2-digit', minute: '2-digit' })
  } else {
    return date.toLocaleDateString('zh-CN')
  }
}

export const truncateText = (text, maxLength) => {
  if (text.length <= maxLength) return text
  return text.substring(0, maxLength) + '...'
}
```

9.2 创建 utils/constants.js

javascript

```javascript
export const API_ENDPOINTS = {
  AUTH: {
    LOGIN: '/auth/login',
    REGISTER: '/auth/register',
    LOGOUT: '/auth/logout',
    REFRESH: '/auth/refresh',
  },
  CHAT: {
    MESSAGES: '/chat/messages',
    ROOMS: '/chat/rooms',
  },
}

export const SOCKET_EVENTS = {
  CONNECT: 'connect',
  DISCONNECT: 'disconnect',
  JOIN_ROOM: 'join_room',
  NEW_MESSAGE: 'new_message',
  USERS_UPDATE: 'users_update',
}
```

第十一步：样式文件

11.1 创建 src/App.css

css

```css
.App {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

/* 登录页面样式 */
.login-container {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}

.login-form {
  background: white;
  padding: 2rem;
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
}

.login-form h2 {
  text-align: center;
  margin-bottom: 1.5rem;
  color: #333;
}

.form-group {
  margin-bottom: 1rem;
}

.form-group input {
  width: 100%;
  padding: 0.75rem;
  border: 1px solid #ddd;
  border-radius: 5px;
  font-size: 1rem;
}

.form-group input:focus {
  outline: none;
  border-color: #667eea;
}
```

```css
.error-message {
  background: #fee;
  color: #c33;
  padding: 0.75rem;
  border-radius: 5px;
  margin-bottom: 1rem;
  text-align: center;
}

button {
  width: 100%;
  padding: 0.75rem;
  background: #667eea;
  color: white;
  border: none;
  border-radius: 5px;
  font-size: 1rem;
  cursor: pointer;
  transition: background 0.3s;
}

button:hover {
  background: #5a67d8;
}

button:disabled {
  background: #ccc;
  cursor: not-allowed;
}

/* 聊天页面样式 */
.chat-container {
  display: flex;
  height: 100vh;
}

.chat-sidebar {
  width: 250px;
  background: #2d3748;
  color: white;
  padding: 1rem;
}

.chat-main {
  flex: 1;
  display: flex;
  flex-direction: column;
}

.chat-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem;
  border-bottom: 1px solid #e2e8f0;
  background: white;
}

.chat-messages {
  flex: 1;
  overflow-y: auto;
  padding: 1rem;
  background: #f7fafc;
}

.message {
  display: flex;
  margin-bottom: 1rem;
  align-items: flex-start;
}

.message-own {
  flex-direction: row-reverse;
}

.message-avatar {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  margin: 0 0.5rem;
  overflow: hidden;
}

.message-avatar img {
  width: 100%;
  height: 100%;
  object-fit: cover;
}

.avatar-placeholder {
  width: 100%;
  height: 100%;
  background: #667eea;
```

```css
  display: flex;
  align-items: center;
  justify-content: center;
  color: white;
  font-weight: bold;
}

.message-content {
  max-width: 70%;
  background: white;
  padding: 0.75rem;
  border-radius: 10px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.message-own .message-content {
  background: #667eea;
  color: white;
}

.message-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 0.25rem;
}

.message-author {
  font-weight: bold;
  font-size: 0.875rem;
}

.message-time {
  font-size: 0.75rem;
  opacity: 0.7;
}

.message-text {
  word-wrap: break-word;
}

.message-input-container {
  display: flex;
  padding: 1rem;
  background: white;
  border-top: 1px solid #e2e8f0;
}
```

```css
.message-input {
  flex: 1;
  padding: 0.75rem;
  border: 1px solid #ddd;
  border-radius: 25px;
  margin-right: 0.5rem;
  outline: none;
}

.message-input:focus {
  border-color: #667eea;
}

.message-input-container button {
  width: auto;
  padding: 0.75rem 1.5rem;
  border-radius: 25px;
}

/* 用户列表样式 */
.user-list h3 {
  margin-bottom: 1rem;
  padding-bottom: 0.5rem;
  border-bottom: 1px solid #4a5568;
}

.user-item {
  display: flex;
  align-items: center;
  padding: 0.5rem 0;
  position: relative;
}

.user-item .user-avatar {
  width: 32px;
  height: 32px;
  margin-right: 0.75rem;
}

.user-name {
  flex: 1;
  font-size: 0.875rem;
}

.user-status {
  width: 8px;
  height: 8px;
  border-radius: 50%;
  margin-left: 0.5rem;
}

.user-status.online {
  background: #48bb78;
}

.user-status.offline {
  background: #a0aec0;
}
```

11.2 创建 src/index.css

css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #f7fafc;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

/* 滚动条样式 */
::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: #f1f1f1;
}

::-webkit-scrollbar-thumb {
  background: #ccc;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: #bbb;
}
```

第十二步：环境配置

12.1 创建 .env 文件

```env
VITE_API_URL=http://localhost:5000
VITE_WS_URL=ws://localhost:5000
```

12.2 修改 vite.config.js 使用环境变量

javascript

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { resolve } from 'path'

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': resolve(__dirname, 'src'),
    },
  },
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: process.env.VITE_API_URL || 'http://localhost:5000',
        changeOrigin: true,
      },
    },
  },
})
```

第十三步：错误处理和优化

13.1 创建 components/ErrorBoundary.jsx

javascript

```
import React from 'react'

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props)
    this.state = { hasError: false, error: null }
  }

  static getDerivedStateFromError(error) {
    return { hasError: true, error }
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error caught by boundary:', error, errorInfo)
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="error-boundary">
          <h2>哎呀，出错了！</h2>
          <p>应用程序遇到了一个错误。</p>
          <details style={{ whiteSpace: 'pre-wrap' }}>
            {this.state.error && this.state.error.toString()}
          </details>
          <button onClick={() => window.location.reload()}>
            重新加载页面
          </button>
        </div>
      )
    }

    return this.props.children
  }
}

export default ErrorBoundary
```

13.2 在 App.jsx 中使用 ErrorBoundary

javascript

```
import ErrorBoundary from './components/ErrorBoundary'

function App() {
  return (
    <ErrorBoundary>
      <Provider store={store}>
        <BrowserRouter>
          <div className="App">
            <Routes>
              {/* 路由配置 */}
            </Routes>
          </div>
        </BrowserRouter>
      </Provider>
    </ErrorBoundary>
  )
}
```

第十四步：性能优化

14.1 使用 React.memo 优化组件

javascript

```javascript
// 在 ChatMessage.jsx 中
import React from 'react'

const ChatMessage = React.memo(({ message, isOwn }) => {
  // 组件代码
})

export default ChatMessage
```

14.2 使用 useCallback 优化事件处理

javascript

```
// 在 MessageInput.jsx 中
import { useCallback } from 'react'

const MessageInput = () => {
  const handleSubmit = useCallback((e) => {
    e.preventDefault()
    // 处理逻辑
  }, [currentRoom, dispatch])

  return (
    // JSX
  )
}
```

15.1 修改 main.jsx

javascript

```javascript
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

15.2 启动项目

bash

```bash
npm run dev
```

重要知识点总结

Redux相关

- **Store配置**: `configureStore`创建store
- **Slice**: 使用`createSlice`管理状态
- **异步操作**: `createAsyncThunk`处理API调用
- **状态选择**: `useSelector`获取状态
- **派发Action**: `useDispatch`派发action

React Router相关

- **路由配置**: `BrowserRouter`, `Routes`, `Route`
- **导航**: `useNavigate`程序式导航
- **路由守卫**: `ProtectedRoute`组件
- **重定向**: `Navigate`组件

Axios相关

- **实例创建**: `axios.create()`
- **拦截器**: 请求/响应拦截器
- **认证**: JWT token处理
- **错误处理**: 统一错误处理

React最佳实践

- **组件拆分**: 功能单一，可复用
- **自定义Hooks**: 封装复杂逻辑
- **状态管理**: 区分本地状态和全局状态
- **副作用处理**: 正确使用useEffect

项目结构

- **分层架构**: 组件、页面、服务、工具分离
- **模块化**: 按功能组织代码
- **可维护性**: 清晰的文件结构和命名

开发建议和进阶学习

1. 逐步实现建议

按以下顺序逐步实现：

1. 先搭建基础架构（Store、路由）
2. 实现登录功能
3. 添加基础聊天界面
4. 集成Redux状态管理
5. 添加实时功能
6. 优化用户体验

2. 调试技巧

- 使用 Redux DevTools Extension 调试状态
- 使用 React Developer Tools 查看组件树
- 在浏览器 Network 面板查看API请求
- 使用 console.log 调试组件渲染

3. 常见问题解决

- **状态更新不及时**: 检查useEffect依赖数组
- **路由跳转失败**: 确认路由配置和认证状态
- **API请求失败**: 检查拦截器和错误处理
- **组件重复渲染**: 使用React.memo和useCallback优化

4. 后续扩展功能

- 文件上传和图片消息
- 消息已读状态
- 用户搜索和添加好友
- 消息加密
- 主题切换
- 国际化支持

这个完整的项目架构为你提供了一个生产级别的React应用基础，所有关键知识点都有详细标注，便于你深入学习和实践。