P2P音乐共享平台需求文档

1. 项目概述

本项目是一个基于WebRTC技术的P2P（点对点）音乐共享平台，允许用户直接在浏览器中共享和下载音乐文件，无需通过中央服务器中转文件数据。

2. 系统架构

- **服务器端**：基于Flask和Socket.IO实现的WebSocket服务器，负责节点连接管理、文件信息共享和WebRTC信令转发
- **客户端**：基于浏览器的Web应用，通过Socket.IO连接服务器，使用WebRTC实现节点间直接通信

3. 核心功能

3.1 文件上传与共享

- 用户可以在本地选择并上传音乐文件
- 上传的文件信息（文件名、大小、类型、唯一ID）会共享到服务器
- 服务器维护所有节点的文件信息列表

3.2 网络文件浏览

- 用户可以查看网络中其他节点共享的音乐文件
- 文件列表会定期自动刷新

3.3 P2P文件传输

- 用户可以直接从其他节点下载音乐文件
- 文件传输不经过服务器，而是通过WebRTC建立的P2P连接直接进行

4. 节点连接与文件传输机制

4.1 节点连接流程

1. **初始化连接**：
    - 客户端打开应用页面后，通过Socket.IO连接到服务器
    - 服务器为每个客户端分配唯一的节点ID（ `client_id` ）
    - 客户端将自己的连接状态显示给用户
2. **文件信息共享**：
    - 客户端上传文件后，将文件元信息（不含实际文件数据）发送到服务器
    - 服务器更新该节点的文件列表，并通知所有其他节点文件列表已更新
    - 其他节点收到通知后，从服务器获取最新的网络文件列表

4.2 文件传输流程

文件传输是本系统的核心，采用WebRTC技术实现P2P传输。详细流程如下：

1. **下载请求发起**：
    - 下载方（请求方）选择要下载的文件，点击"下载"按钮
    - 客户端向服务器发送下载请求，包含目标节点ID和文件ID

```
socket.emit('request-download', {
  targetNodeId: nodeId,
  fileId: fileId
});
```

2. **连接建立与信令交换**：
    - 服务器将下载请求转发给目标节点
    - 下载方开始建立P2P连接：
        - 创建 `RTCPeerConnection` 对象，配置STUN服务器用于NAT穿透
        - 创建数据通道(`DataChannel`)用于文件传输
        - 生成连接offer并发送给目标节点
    - 目标节点收到offer后：
        - 创建 `RTCPeerConnection` 对象
        - 监听数据通道事件

        - 生成answer并发送回下载方
    - 双方交换ICE候选信息，建立直接连接

3. **文件数据传输**：

    - 连接建立后，目标节点（文件提供方）通过数据通道发送文件：
        - 先发送文件元数据（文件名、大小、类型等）
        - 将文件分割成多个数据块（chunk）依次发送
        - 监控数据通道缓冲区状态，避免过度缓冲
        - 发送完成后，发送一个特殊的完成标记
    - 下载方接收文件数据：
        - 接收并解析文件元数据
        - 接收文件数据块并存储在缓冲区
        - 实时显示接收进度
        - 接收完成后，合并所有数据块

4. **文件处理与下载**：

    - 下载方将接收到的数据合并成完整文件
    - 创建Blob对象和Object URL
    - 自动触发文件下载，使用原始文件名
    - 下载完成后清理资源，关闭数据通道

5. 技术栈

- **服务器端**：

    - Python
    - Flask
    - Flask-SocketIO
    - eventlet（异步服务）

- **客户端**：

    - HTML5
    - JavaScript
    - Socket.IO客户端
    - WebRTC API

- **网络服务**：
    - STUN服务器（Google公共STUN服务器）用于NAT穿透

6. 特殊功能与设计考量

6.1 NAT穿透

系统使用Google公共STUN服务器（ `stun:stun.l.google.com:19302` 和 `stun:stun1.l.google.com:19302` ）帮助节点穿透NAT，建立直接连接。

6.2 数据可靠性保障

- 文件传输采用有序数据通道（ `ordered: true` ）确保数据块按顺序到达
- 实现了文件传输完成标记机制，确保接收方能够正确识别传输结束
- 数据合并和Blob创建过程有完善的错误处理机制

6.3 性能优化

- 文件分块传输，避免一次性加载大文件到内存
- 监控数据通道缓冲区状态，避免过度缓冲
- 采用延迟关闭连接和资源释放策略，确保文件传输完整

6.4 用户体验优化

- 实时显示连接状态和文件传输进度
- 自动触发文件下载，无需用户手动保存
- 提供友好的通知提示

7. 使用方法

1. 启动服务器： `python server.py`
2. 通过浏览器访问显示的链接（本地访问： `http://localhost:3000` ）
3. 上传音乐文件到平台
4. 浏览网络中其他用户共享的音乐文件
5. 点击"下载"按钮直接从其他节点下载音乐文件

8. 注意事项

- 系统仅支持音频文件共享
- 文件传输需要双方都保持在线
- 在某些复杂网络环境下，可能无法建立P2P连接