QTune 项目报告

组员：沈哲语 陈旃旎 段可峥

一、程序功能介绍

视唱练耳小助手Qtune，主要面向音乐学习者，帮助用户进行视唱、听音、写谱等音乐基础训练。

- 视唱（Sing）：分为简单、中等、困难三个等级。
  - 自动生成乐谱：根据不同难度自动生成乐谱。点击“下一个”进行多轮练习。
  - 录音、评分与回放：用户可以跟着乐谱进行视唱并用麦克风录音，系统会对用户的演唱进行自动评分，录音后可回听自己的演唱，也可收听正确答案。
- 听音（Listen）：分为单音、双音音程、三和弦、七和弦的训练。
  - 听辨训练：可以播放不同的训练音频，支持下载wav文件（可导入手机音乐软件中反复收听，达到磨耳朵的作用）。
  - 自测：自动根据不同板块生成考题，先播放国际标准音A，然后再播放考题。用户输入答案后，系统会判断正误，并给出正确答案。
- 写谱（Write）
  - MIDI文件转乐谱：可以解析用户上传的MIDI文件并生成乐谱图片。
  - 录音转乐谱：用户也可以直接录音，系统通过对录音进行音高分析来自动生成乐谱，即可帮助用户快速扒旋律。
  - 乐谱图片导出：自动生成的乐谱可以保存为图片，便于打印或分享。
- 日志（Log）：
  - 学习成果可视化：自动统计并显示总练习天数，直观展示坚持与努力；通过动态饼图，一目了然地展示视唱练习中的总平均分与听音训练里的总正确率，快速掌握自己的综合水平。
  - 多维度练习趋势分析：为“视唱”和“听音”两大模块提供独立的趋势分析图表，折线图追踪每日平均分/正确率的变化，帮助发现进步或瓶颈；柱状图记录每日练习次数，反映练习投入度。
  - 一键重置：提供数据重置功能，可以清空所有历史练习记录，方便从新开始或进行数据清理。
- AI助手（AI Assistant）：
  - 集成专业大语言模型： 内置专门为音乐理论和视唱练耳优化的AI助手（基于DeepSeek API），能够解答学习疑惑。
  - 上下文连续对话：支持多轮对话，AI能够理解上下文语境。可以就一个问题进行深入追问和探讨，获得更连贯、精准的解答。
  - 实时流式响应： 问题提交后，AI的回答会像打字一样逐字生成并实时展示，无需漫长等待，交互体验流畅自然。
  - API密钥本地保存：用户只需输入一次自己的API Key，程序便会自动加密并保存在本地，无需每次启动都重新输入，方便快捷。
- 界面：多窗口切换，各功能模块有独立窗口，支持返回上一层等操作。

二、项目各模块与设计细节

（1）主界面与导航（Homepage）

- `Homepage` 类（homepage.h/.cpp），继承自 `QWidget`，是程序的主窗口。内部持有 Sing、Listen、Write、Log 四个功能模块的窗口指针，负责模块切换。
- 通过按钮点击事件，隐藏主界面并显示对应功能窗口。
- 负责主界面美化与按钮样式设置。

（2）视唱模块（Sing）

Sing 类 (sing.h, sing.cpp)

- 基类: `QWidget`，负责视唱功能的入口界面与内容准备。
- 主要成员:
  - `SingSub* singsub`: 持有对核心练习界面 `SingSub` 的指针，用于管理界面的切换与数据传递。
- 实现细节:
  1. 难度选择: 界面提供“简单”、“中等”、“困难”三个按钮。当用户点击任一按钮时，会触发相应槽函数（如 `on_easybtn_clicked`）。
  2. 内容生成: 在槽函数内部，首先调用 `ChordGenerator` 类的方法，根据所选难度在内存中动态创建一个 `Score` 对象。此对象是乐谱的抽象数据模型，包含了所有音符的音高和时值信息。
  3. 多媒体资源准备: 接着，程序利用此 `Score` 对象同步生成两种核心资源：
     - 调用 `drawScore()` 函数，将 `Score` 对象渲染成一张PNG格式的乐谱图片（`tmp.png`），供用户视觉识别。
     - 调用 `scoreToWAV()` 函数，将 `Score` 对象合成为一个WAV格式的标准音频文件（`tmp.wav`），作为听觉参考。
  4. 所有资源准备完毕后，`Sing` 界面会隐藏自身，并调用 `singsub->show()` 来显示练习界面。同时，将生成的 `Score` 对象和表示难度的整数值传递给 `SingSub` 实例，以便其进行后续操作。

SingSub 类 (singsub.h, singsub.cpp)

- 基类: `QWidget`，负责核心的视唱练习与评分工作。
- 主要成员:
  - `QAudioSource* audioInput` / `QFile* audioFile`: 用于管理麦克风输入并将音频流写入文件。
  - `QMediaPlayer* player` / `QAudioOutput* audioOutput`: 用于播放所有音频，包括用户的录音回放和标准答案。
  - `Score testScore`: 存储由 `Sing` 类传递过来的标准乐谱数据。
  - `Score audioScore`: 用于存储从用户录音中分析重建出的乐谱数据。
  - `int currentScore`: 保存最终计算出的练习分数。
- 实现细节:
  1. 录音与WAV文件封装:

- 当用户点击“开始录音”，`startRecording()` 被调用。它会初始化 `QAudioSource`，并将音频流直接写入一个本地文件（`record.wav`）。
- 录音开始时，会先调用 `writeWavHeader()` 向文件写入一个包含占位符的WAV文件头。
- 当录音结束时，`stopRecording()` 被调用。此时会调用 `updateWavHeader()`，用真实的文件大小信息更新文件头，从而生成一个格式完整的WAV文件。

2. 智能评分算法 (`scoreRecording`): 在录音结束后自动执行。
   - 音频解码: 读取 `record.wav` 文件，并将其PCM数据解码为标准化的单声道浮点数数组。
   - 特征提取: 使用 `Gist` 音频分析库，以帧为单位遍历音频数据，提取出每一帧的音高（基频）和能量。
   - 音符重建: 通过分析音高和能量的连续性与变化，系统能从连续的音频信号中识别并切分出离散的音符事件（包含起始时间、时长和平均音高），并用这些音符构建出 `audioScore` 对象。
   - 比对评分: 调用 `scorePerformance()` 函数，将用户演唱的 `audioScore` 与标准的 `testScore` 进行逐音符比对。通过评估音准和节奏的匹配度，计算出最终的百分制分数并更新 `currentScore`。
3. 结果持久化 (`saveScoreToFile`):
   - 当一次有效的练习（即用户录了音并获得了分数）结束时，该函数会被调用。
   - 它会将本次练习的日期时间、难度等级和最终分数格式化成一个字符串，并以追加模式写入到日志文件 `log_file/scores_log.txt` 中，为后续数据分析提供支持。

（3）听音模块（Listen）

Listen 类 (listen.h, listen.cpp)

- 基类: `QWidget`，负责听音功能的入口界面与内容准备。
- 主要成员:
  - `ListenSub* listensub`: 持有对核心练习界面 ListenSub 的指针，用于管理界面的切换与数据传递。
- 实现细节:
  1. 难度选择: 界面提供"单音"、"双音"、"三音"、"七音"四个按钮。当用户点击任一按钮时，会触发相应槽函数（如 `on_danyin_clicked`）。
  2. 内容生成: 在槽函数内部，调用 `ChordGenerator` 类的方法，根据所选难度在内存中动态创建一个 `Score` 对象，用于 `listensub` 中的收听。（因为谱子遍历整个八度，生成时间较长，所以为了减少加载时间，我们将谱子和后续生成的图片和音频存在本地文件夹中供以后访问使用。即，第一次生成完成后，以后就会略过生成的步骤，在 `listensubsub` 中直接使用已生成的本地资源）
  3. 多媒体资源准备: 接着，程序利用此 Score 对象同步生成两种核心资源：
     - 调用 `drawScore()` 函数，将 Score 对象渲染成一张PNG格式的乐谱图片（`tmp.png`），供用户视觉参考。
     - 调用 `scoreToWAV()` 函数，将 `Score` 对象合成为一个WAV格式的标准音频文件（`tmp.wav`），作为听觉训练。
     - 所有资源准备完毕后，Listen 界面会隐藏自身，并调用 listensub->show() 来显示练习界面。同时，通过 `setMode()` 方法将表示难度的整数值传递给 ListenSub 实例。

ListenSub 类 (listensub.h, listensub.cpp)

- 基类: `QWidget`，负责听音练习的核心界面与音频播放功能。
- 主要成员:
  - `QMediaPlayer* player` / `QAudioOutput* audioOutput`，用于播放训练音频文件。
  - `ListenSubSub* listensubsub`: 持有对测试界面 `ListenSubSub` 的指针，用于管理界面的切换。
  - int mode: 存储当前选择的难度模式（1-4分别对应单音、双音、三和弦、七和弦）。
- 实现细节:
  1. 乐谱显示 `showYuepu()`: 根据当前难度模式，从对应的训练文件夹中加载乐谱图片（tmp.png），并进行等比例缩放以适应界面显示。
  2. 音频播放控制 `on_play_clicked()`: 根据当前难度模式，从对应的训练文件夹中加载音频文件（tmp.wav）。使用 `QMediaPlayer` 实现播放/暂停功能，并同步更新播放按钮的状态。
  3. 文件下载功能 `on_download_clicked()`: 允许用户将当前训练的音频文件保存到本地指定位置。
  4. 测试功能 `on_test_clicked()`: 停止当前播放，隐藏自身界面，并调用 `listensubsub->show()` 显示测试界面，同时通过 `setsubMode()` 方法传递难度信息。

ListenSubSub 类 (listensubsub.h, listensubsub.cpp)

- 基类: `QWidget`，负责听音测试与答案验证功能。
- 主要成员:
  - `QMediaPlayer* player` / `QAudioOutput* audioOutput`: 用于播放测试音频文件。
  - `QString answer`: 存储当前测试的标准答案。
  - int mode: 存储当前测试的难度模式。
- 实现细节:
  1. 动态测试生成 `on_play_clicked()`:
     - 当用户首次点击播放按钮时，调用 `ChordGenerator` 类的 `generateExam(mode)` 函数动态随机生成一个测试用的 Score 对象和对应的标准答案。
     - 将生成的乐谱转换为WAV音频文件（`tmp.wav`）并设置给播放器。用户即可通过点击播放收听测试音频。
     - 将标准答案存储在 `answer` 变量中，并在界面上显示。为了隐藏答案，先用一个覆盖标签遮挡。
  2. 答案验证 `checkAnswer()`:
     - 当用户按下回车键或提交答案时，将用户输入的答案与标准答案进行字符串比较（忽略大小写和首尾空格）。
     - 在界面上显示"正确"或"错误"的结果，并取消覆盖标签的遮挡以显示正确答案。
     - 调用 `logListenEvent()` 记录测试结果，将本次测试的时间戳、难度模式和结果（正确为1，错误为0）格式化成一个字符串。以追加模式写入到日志文件 log_file/listen_log.txt 中，为后续数据分析提供支持。
     - 事件过滤 (`eventFilter`): 监听用户输入框的键盘事件，当检测到回车键时自动触发答案验证。

3. 界面重置 `on_refresh_clicked()`:
    - 清除当前测试结果和用户输入。停止音频播放并清空播放器源。重置答案变量并显示覆盖标签。重新启用输入框，准备下一次测试。重新生成随机试题。

（4）写谱模块（Write）

Write 类 (write.h, write.cpp)

- 基类: `QWidget`，该类负责接收用户的输入（MIDI文件或麦克风录音），执行核心的转换和分析任务，并协调与子窗口的交互。
- 主要成员:
    - `WriteSub* writesub`: 指向结果展示子窗口的指针。
    - `QAudioSource* audioInput` / `QFile* audioFile`: 用于管理麦克风录音及文件写入。
    - `QMediaPlayer* player` / `QAudioOutput* audioOutput`: 用于回放录制的音频。
    - `Score midiScore` / `Score audioScore`: 分别用于存储从MIDI文件和录音中解析出的乐谱数据结构。
    - `QString currentMidiPath` / `QString currentSoundPath`: 存储当前处理的MIDI文件或录音文件的路径。
    - `bool hasMidi`, `bool hasSound`, `bool isRecording`: 用于管理界面状态的状态标志。
- 实现细节:
    1. 双重输入模式: 该模块支持两种工作流，用户可以选择其一进行操作。
        - MIDI文件转乐谱: 用户通过“选择midi文件”按钮加载一个本地的 `.mid` 文件。当用户点击“开始处理”时，程序会调用 `parseMidiToScore` 函数。该函数利用 `MidiFile` 第三方库，读取MIDI文件的事件序列，最终将所有音符信息（音高、起止时间等）存入 `midiScore` 对象中。
        - 录音转乐谱: 用户通过“开始录音”按钮启动麦克风录制。音频流被实时写入一个本地的WAV文件（写入过程与Sing模块类似）。当用户点击“开始处理”后，程序进入音频分析流程。
    2. 音频转写算法 (`processAudio`):
        - 解码与准备: 程序首先读取录制的WAV文件，将其音频数据解码为标准化的单声道浮点数数组，这是后续信号处理的基础。
        - 特征提取: 利用 `Gist` 音频分析库，程序以帧为单位对音频数据进行处理，提取出每一帧的音高（基频）和能量信息。
        - 音符切分与重建: 系统通过一套基于能量和音高稳定性的规则，从连续的音频信号中识别出离散的音符。当能量超过静音阈值且音高稳定时，标记为一个音符的开始；当能量回落或音高发生剧烈跳变时，标记为音符的结束。此过程将用户的演唱重建为 `audioScore` 对象。为了提高转写质量，算法还会对分析出的相邻且音高相同的短音符进行合并，形成更连贯的音符。
    3. 统一的输出流程: 无论输入是MIDI还是录音，处理流程的最后一步是相同的。程序会调用 `drawScore` 函数，将最终得到的 `Score` 对象（ `midiScore` 或 `audioScore` ）渲染成一张PNG格式的乐谱图片。随后，该图片被传递给 `writesub` 子窗口进行显示。

WriteSub 类 (writesub.h, writesub.cpp)

- 基类: `QWidget`，该类仅用于显示生成的乐谱图片，并提供保存功能。
- 主要成员:
  - `QImage resultImage`: 存储由`Write`类传递过来的乐谱图片。
- 实现细节:
  1. 图片显示 (`setResultImage`): 该函数由`Write`类调用，用于接收生成的乐谱图片。它会将传入的`QImage`对象转换为`QPixmap`，并根据界面控件的大小进行等比例缩放，以保证乐谱清晰且完整地显示，最后将其设置到一个`QLabel`控件上。
  2. 图片导出 (`on_download_clicked`): 用户点击“下载”按钮后，该函数会弹出一个文件保存对话框（`QFileDialog::getSaveFileName`），允许用户选择保存路径和文件名。一旦用户确认，程序便会调用`resultImage.save()`方法，将内存中的乐谱图片以用户选择的格式（如PNG, JPG）保存到本地磁盘。

（5）日志模块（Log）

Log 类 (log.h, log.cpp)

- 基类: `QWidget`，负责从本地日志文件中读取用户的练习历史，并将这些数据处理、聚合后，通过多种图表形式直观地展示给用户。
- 主要成员:
  - `Ui::Log *ui`: 指向UI界面的指针，其中包含了用于显示图表的 `QChartView` 控件和显示统计数字的标签。
  - `QString logPath`: 存储日志文件所在目录的路径。
  - `QMap<QDate, QMap<int, QList<int>>>`: 这是模块核心的在内存中的数据结构，用于存储从文件中读取的练习数据。第一层 `QMap` 以日期为键，第二层以练习模式（如难度等级）为键，`QList` 则存储该日该模式下的所有成绩或结果。
- 实现细节:
  1. 动态加载与刷新: 为了确保用户总能看到最新的数据，模块的核心逻辑被放置在 `showEvent` 事件处理器中。每当日志窗口被显示时，都会自动触发 `loadAndDisplayLogs` 函数，该函数会重新读取所有日志文件并刷新界面上的所有图表和数据。
  2. 日志文件解析:
     - 模块定义了三个文件读取函数（`readLoginData`, `readListenLog`, `readScoresLog`），分别对应三个日志文件 (`date.txt`, `listen_log.txt`, `scores_log.txt`)。
     - 这些函数以只读文本模式打开文件，逐行读取内容。每一行都被解析为一条独立的练习记录，程序从中提取出关键信息，如日期时间、练习模式和分数/结果。
     - 解析出的数据被有序地存入前述的嵌套 `QMap` 数据结构中，以便进行高效的查询和聚合。
  3. 数据聚合与图表生成: 这是模块的核心功能，通过一系列 `setup...` 函数实现。
     - 整体能力总览 (`setupPieCharts`):
       - 此函数遍历所有已加载的视唱和听音数据，计算出用户有史以来的“总平均分”（视唱）和“总正确率”（听音），使用 `QPieSeries` 来创建两个饼图。
     - 近期练习趋势分析 (`setupScoresTrendChart`, `setupListenTrendChart`):

        - 这两个函数的功能是创建复杂的组合图表，用于展示最近7天的练习趋势。
        - 数据处理: 它们会遍历从今天起过去7天的每一天。对于每一天，程序会从内存数据中聚合出当日各个模式下的“日均分/正确率”和“日练习次数”。
        - 图表构建: 为了在同一张图上展示两种不同度量单位的数据（百分比和次数），图表采用了双Y轴设计。左侧Y轴表示分数或正确率（0-100），右侧Y轴表示练习次数。用户的表现（平均分/正确率）通过折线图（`QLineSeries`）展示，而练习投入度（次数）则通过柱状图（`QBarSeries`）展示。X轴则以日期（月-日）作为分类标签。
    4. 数据管理 (`on_resetButton_clicked`, `clearLogFiles`):
        - 模块提供了一个“重置数据”的功能。当用户点击此按钮并确认后，`clearLogFiles` 函数会被调用。
        - 该函数会以写入并截断（`QIODevice::Truncate`）的模式打开所有日志文件，此操作会立即清空文件内容，从而实现所有练习数据的重置。重置后，界面会自动刷新，显示为空白状态。

（6）AI助手（AI Assistant）

AIAssistant 类 (aiassistant.h, aiassistant.cpp)

- 基类: `QWidget`
- 与第三方大语言模型（LLM）API交互的客户端。该模块封装了网络通信、JSON数据处理、对话历史管理以及API密钥的本地持久化，为用户提供一个专业的音乐学习问答机器人。
- 主要成员:
    - QNetworkAccessManager* m_networkManager: Qt的网络访问核心类，用于发起和管理网络请求。
    - QNetworkReply* m_currentReply: 指向当前正在进行的网络应答对象，方便对其中断和管理。
    - QJsonArray m_chatHistory: 存储整个对话的上下文。每一条消息（包括系统提示、用户输入和AI回复）都作为一个JSON对象存储在该数组中，以实现连续对话。
- 实现细节:
    1. 网络通信与API交互:
        - 当用户点击“发送”按钮时，on_sendButton_clicked 函数被触发。它首先会校验API Key和用户输入是否为空。
        - 接着，它会构建一个HTTP POST请求，目标地址为DeepSeek API的聊天接口。请求头中包含了必要的认证信息（Bearer Token）和内容类型（application/json）。
        - 请求体是一个JSON对象，其中包含了模型名称（如deepseek-chat）、完整的对话历史m_chatHistory，以及一个关键参数"stream": true，用于请求流式响应。
        - 请求通过m_networkManager->post()发出，并连接readyRead和finished信号到相应的槽函数，以异步方式处理网络响应。
    2. 流式响应处理 (onReplyReadyRead):
        - 为了提供流畅的交互体验，模块采用了流式（Streaming）API。这意味着服务器会持续地发送小块数据（data chunks），而不是等待整个回答生成完毕后一次性返回。
        - onReplyReadyRead 槽函数在每次接收到新数据时被调用。它会解析遵循SSE（Server-Sent Events）协议的数据流。

- 每一行以 "data: " 开头的数据块都包含一个JSON对象，其中含有AI生成的增量文本（deltaContent）。程序会提取这部分文本，并立即追加到界面的文本框中，实现打字机般的显示效果。

3. 上下文管理 (m_chatHistory):
   - 为了让AI能够理解对话的上下文，模块维护了一个m_chatHistory数组。
   - 在程序启动时，会首先向历史记录中加入一条“系统”角色的消息，为AI设定其专业领域（音乐理论助手），这有助于引导AI生成更相关的回答。
   - 每次用户发送新问题时，该问题会作为“用户”角色的消息被追加到历史记录中。
   - 当一次完整的AI流式响应结束后，在onReplyFinished函数中，程序会将整个AI的回答拼接起来，作为“助手”角色的消息追加到历史记录中。这样，下一次请求就会包含完整的对话历史，实现上下文连续性。
4. API密钥的持久化:
   - 为了方便用户，模块实现了API Key的本地保存功能。
   - saveApiKey: 当用户离开该界面或关闭程序时，此函数被调用。它会获取输入框中的API Key，使用Base64进行简单的编码，然后将其写入程序目录下的config.txt文件中。编码可以防止密钥以明文形式直接暴露在文件中。
   - loadApiKey: 程序启动时调用此函数。它会检查config.txt文件是否存在，如果存在，则读取内容，进行Base64解码，并将解码后的Key设置回输入框，免去了用户每次启动都需重新输入的麻烦。

(7) 乐谱与音符数据结构

- Note 类（note.h）

  表示单个音符，包含音高、力度、起止时间、MIDI 通道、在谱面上的位置等属性。提供音符名、升降号、时值等获取函数。

- Score 类（note.h）

  表示一段乐谱，由多个 Note 组成。提供添加、清空、合并等操作。

(8) 乐谱生成与音频分析

- ChordGenerator 类（generatescore.h/.cpp）

  负责生成不同难度、不同类型的乐谱（如视唱、和弦等）。支持多种和弦、节奏、音型模式。

- `Score` 对象转图片 (drawscore.h/.cpp):
  - 定义了谱线间距、音轨间距、音符宽度、音符大小、升降号大小、页边距、页宽等常量。可以进行调整以达到美观的效果。
  - 绘制采用 `QPainter` 直接绘制。先计算出乐谱大小，画直线作为谱线，标记谱号和音轨。再通过音符发生时间和音符音高计算出画面上的位置。然后调用`note`类的 `getSymbol()` 获得音符时值对应的符号，调用`note`类的 `getAccidentSymbol()` 获得变音符号，放置在先前计算出的位置。当音符超出五线谱范围时，自动添加加线。
  - 提供了 `drawScore(Score& score, const QString& outputDir)` 方法，打包了所有绘制步骤以及保存步骤（将`QImage` 对象保存为PNG格式文件）。调用时只需传入待转为图片的 `Score` 对象以及希望储存的地址即可。
- `Score` 对象转音频 (playscore.h/.cpp)

    - 音频转换采用 FluidSynth 软件合成器。先将 `Score` 对象转换为标准MIDI文件格式，包括文件头、音轨数据和事件序列。然后调用FluidSynth命令行工具，配合GeneralUser-GS音色库，将MIDI文件渲染为高质量的WAV音频文件。
    - 提供了 `scoreToWAV(Score& score, const QString& outputPath)` 方法，封装了完整的音频生成流程。该方法内部先调用 `scoreToMIDI()` 生成临时MIDI文件，再通过FluidSynth转换为WAV格式，最终在指定目录下生成 `tmp.wav` 文件。调用时只需传入待转换的 `Score` 对象和输出目录路径即可。
- Gist/Yin/GistFFT 等类（Gist/ 目录下）

  用于音频信号分析，如音高检测、特征提取等。

  主要用于录音转乐谱、自动评分等功能。

（9）UI设计

各模块均有对应的 .ui 文件，采用 Qt Designer 设计

资源文件picture.qrc管理贴图资源，包括矢量图、背景图

控件美化主要通过修改样式表实现

部分ui控件采用矢量图贴图的形式实现美化

界面切换通过多窗口实现，各功能模块有独立窗口，支持返回上一层等操作

三、小组成员分工情况

- 沈哲语：
    - 写谱（Write）功能的后端实现
    - 视唱（Sing）评分功能的实现
    - 日志（Log）功能的前端和后端实现
    - AI助手（AI Assistant）功能的前端和后端实现
- 陈旃旎：
    - Note 和 Score 类的实现（note.h/.cpp）
    - ChordGenerator 类的实现（generatescore.h/.cpp）
    - `Score` 对象转图片的实现（drawscore.h/.cpp）
    - `Score` 对象转音频的实现（playscore.h/.cpp）
    - 结合以上后端程序实现了前端视唱（Sing）和练耳（Listen）的部分功能
- 段可峥：
    - Homepage功能前端的实现
    - Write功能前端的实现
    - Sing功能前端的实现
    - Log功能前端的实现
    - Listen功能前端的实现
    - AI Assistant功能前端的实现

四、项目总结与反思

- 我们的项目一期工程从4.27开始，5.31结束，并参加了路演，二期工程从6.23开始，6.30结束，期间共计召开6次组会，最终实现了一个功能比较完备的试唱练耳软件。
- 沈哲语：

  我本人艺术造诣不高，因此主要负责音频录音、识别、转写、评分，日志，AI接入等几个方面的工作。

  诚如助教所言，音频的处理相对比较特别，因此实现过程中也遇到了不少的困难，期间确实也有过放弃的想法，但在多次尝试之后，最终也是成功实现了原计划的功能。在此期间，也深入认识了音频文件如midi,wav等的结构，音符事件识别的原理，也学会了如何导入外部库和使用cmake,qmake，事实上这并非想象的那么容易。在实现日志的功能时，统计的数据来自于程序的各个部分，最终汇总为图表。主要是实践了数据处理的一些朴素方法，并学到了QTCharts的使用。AI功能则源自于我充的DeepSeek API 没有用完，于是提出了接入AI的想法。在实现的过程中，学到了如何实现网络通信操作。

  当然，限于个人能力，录音转乐谱的功能还会受到一些噪音的干扰，因此后续也可以在这方面作一些降噪的尝试以提升健壮性。另外遗憾的是，到了项目的尾声，本人才意识到Github在管理代码上是相当好用的，只能算是以后的项目积累一些经验了。

  最后，我此前也没有过像这样大规模项目的合作经验，基本是从零开始，当然最后的结果也是相当满意的。这学期结识了两位能力很强，活跃且有想法的队友，合作体验良好，除了项目本身以外也进行了一些课程学业上的交流，一起打了PKUCPC比赛等等，在此也要感谢两位队友一学期以来的合作和支持！

- 陈旃旎：

  首先要衷心感谢我的两位队友。这次合作体验非常愉快，我十分享受和大家一起完成这个项目的过程。我们分工明确，每个人都积极负责地完成自己的任务，遇到困难时互帮互助，彼此信任。平时工作时的交流也很轻松愉快，没想到树洞捞到的队友可以这么合拍！真的很幸运。
  因为“视唱练耳小助手”算是我的个人需求，所以还要特别感谢两位队友愿意一起来写！因为现在市面上没有免费好用的视唱练耳软件，所以这次的大作业算是完成了我的一个心愿吧！
  通过这次项目，我完整经历了从零开始开发一个程序的全过程。从对QT框架一无所知，到能够运用它实现各种功能，我学到了不少东西。我也体会到了如何将需求转化为实际功能，并在开发过程中不断调整优化。看着我们三个人最终完成了一个功能完整的程序，其实是有些惊喜的，这也让我对我们现阶段的码力有了更多的信心。
  总之这学期的程设课，尤其是大作业，是很好的体验！祝程设越办越好！！

- 段可峥：
  一次非常棒的小组作业体验，大家都非常靠谱，每个人都积极完成自己的任务，整个项目完成期间没有任何的拖沓，真的非常感谢两位队友！大家都很棒！

  软件的write功能中音频转谱这个想法是我提出的，这也源于我自己的需求——平时在写曲的时候想要扒谱但是找不到好用的软件，很多时候只能自己手扒。write实现了给出音频就可以给出一个较为准确的谱子，略加修改之后就可以实现将这个音频扒谱出来，非常的方便。

  这次项目中我负责前端的搭建和美化，从零开始学习如何做一个软件前端是一个非常有趣的过程，过程中逐渐了解到平时使用的软件中诸如页面切换、功能按键等是如何通过代码实现的，也了解到软件中界面布局、贴图等美化是如何实现的。因为自己的美术水平不高，所以做的时候比起敲代码更让我头大的是做美化（），每个页面的设计都绞尽脑汁，最后也是将一整个软件的美化这样做了出来，不过这样做一整套关联界面的美化的过程还是非常有趣的。做后端的两个同学也非常靠谱，虽然我前端开始写的比较乱，一些变量命名的也奇奇怪怪，但是两个同学很好的在这个基础上搭好了很棒的后端，真的非常感谢两位同学！

  感谢程设课堂为我带来这么棒的团队合作体验！也感谢两位队友这学期以来的帮助与支持！