TOHEAR

张皖男(2212190106)

徐显舜(2212190527)

一、项目介绍 [张皖男、徐显舜]

1.1 背景与问题陈述

在当今信息快速发展的时代，播客成为一种受欢迎的内容传播方式。传统音频平台存在内容零散、交互单一、个性化推荐不足等问题。

本项目旨在开发一个基于前后端分离的移动端播客平台 —— **ToHear 听书小程序**，为用户提供音频收听、内容收藏、评论互动等功能，提升音频内容获取的效率与体验。

1.2 项目目标与价值

本项目的目标是打造一个简洁、流畅、功能完备的播客收听与分享平台。其核心价值包括：

- 提供高效的音频内容获取与播放体验
- 支持用户注册、登录、收藏、评论等交互功能
- 支持后端内容管理与数据分析，便于系统维护和优化
- 面向移动端设计，提升用户便捷使用体验

1.3 功能需求分析

功能性需求

1. **用户管理功能**
   - 用户注册、登录、登出
   - 用户个人信息查看与编辑
2. **播客内容管理**
   - 播客音频上传、删除、分类管理
   - 播客内容的浏览、收藏、点赞、评论
   - 播客播放记录、历史记录保存
3. **搜索与推荐**
   - 支持关键词搜索播客
   - 首页推荐热门播客/新上架内容
4. **社交互动**
   - 播客评论功能
   - 用户关注/取消关注功能

非功能性需求

- **性能要求**
  - 页面平均响应时间小于 1s；

移动端特殊需求

- **低功耗优化**
  - 后台播放优化电池使用；

1.4 技术选型与架构概述

技术选型

本项目采用前后端分离架构，前端基于跨平台框架 uni-app 开发，后端使用 Spring Boot 构建 RESTful API，数据库采用 MySQL，整体架构清晰、易于扩展与维护。

层级	技术	理由
前端	uni-app（Vue3）	支持跨平台开发，生态完善
后端	Spring Boot	结构清晰，开发效率高
数据库	MySQL	关系型数据库，适合结构化数据
工具	HBuilderX / IDEA / Navicat	前后端协作开发工具

前端架构

- 使用框架：uni-app
- 项目结构划分：

```
tohear_clent/
├── vite/                    # Vite构建工具
├── node_modules/            # 项目依赖
└── src/                     # 源代码目录
    ├── assets/
    │   └── iconfont.css     # 图标字体
    ├── components/          # 公共组件
    │   ├── broadcast-item/  # 广播组件
    │   ├── channel-item/    # 频道组件
    │   ├── comment-item/    # 评论组件
    │   ├── drop-down/       # 下拉组件
    │   ├── edit-wind/       # 编辑窗口
    │   ├── head-generic/    # 通用头部
    │   ├── my-button/       # 自定义按钮
    │   ├── my-head/         # 个人中心头部
    │   ├── player-bar/      # 播放器控制条
    │   ├── popupwindow/     # 弹窗组件
    │   ├── post-head/       # 帖子头部
    │   ├── post-interaction/# 帖子交互
    │   ├── post-item/       # 帖子项
    │   ├── reply-item/      # 回复项
    │   ├── search-box/      # 搜索框
    │   ├── sort-method-box/ # 排序选择
    │   ├── tabbar/          # 底部导航
    │   ├── tags-tabs/       # 标签页
    │   └── useritem/        # 用户项
    ├── pages/               # 页面目录
    │   ├── broadcast/       # 广播页
    │   ├── channel/         # 频道页
    │   ├── community/       # 社区页
    │   ├── creation/        # 创作页
    │   ├── index/           # 首页
    │   ├── login-register/  # 登录注册
    │   ├── my/              # 个人中心
    │   ├── player/          # 播放器
    │   ├── search-pages/    # 搜索页
    │   ├── test/            # 测试页
    │   └── user/            # 用户页
    ├── request/             # 请求封装
    │   ├── apis.ts          # API接口
    │   └── index.ts         # 请求核心
    ├── static/              # 静态资源
    │   └── tab_icons/       # 标签图标
    │       ├── bg.png       # 背景图
    │       └── logo.png     # Logo
    ├── stores/              # 状态管理
    │   ├── base.ts          # 基础Store
    │   ├── counter.ts       # 计数器Store
    │   ├── player-副本.ts   # 播放器备份
    │   ├── player.ts        # 播放器状态
    │   └── user.ts          # 用户状态
    ├── styles/              # 样式文件
    │   └── tailwind.css     # Tailwind样式
    ├── uni_modules/         # uni-app模块
    ├── utils/               # 工具函数
    │   ├── broadcastSort.ts # 广播排序
    │   ├── channelSort.ts   # 频道排序
    │   ├── checkPassword.ts # 密码验证
    │   ├── commentSort.ts   # 评论排序
    │   ├── durationFormatting.ts # 时长格式化
    │   ├── initPlayer.ts    # 播放器初始化
    │   ├── logOut.ts        # 登出逻辑
    │   ├── numConversion.js # 数字转换
    │   ├── postSort.ts      # 帖子排序
    │   └── timeChange.ts    # 时间转换
    ├── App.vue              # 根组件
    ├── env.d.ts             # 环境类型
    ├── main.ts              # 主入口
    ├── manifest.json        # 应用配置
    ├── pages.json           # 路由配置
    ├── shime-uni.d.ts       # uni类型声明
    ├── uni.scss             # 全局样式
    ├── .gitignore           # Git配置
    └── index.html           # HTML入口
```

TypeScript 规范

- 样式规范：
  - 响应式布局，适配各类手机屏幕
  - 统一颜色与字体风格（主题色、组件风格统一）

2.2.2 后端架构

- 技术栈：Spring Boot + MyBatis Plus
- 项目结构：

```
tohear_server/
├── .idea/                    # IDEA配置文件
├── .mvn/                    # Maven包装器
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com.pxx.ifmserver/
│   │   │       ├── config/          # 配置类
│   │   │       │   ├── CorsConfig   # 跨域配置
│   │   │       │   └── WebConfig    # Web配置
│   │   │       ├── controller/      # 控制器层
│   │   │       │   ├── BroadcastController
│   │   │       │   ├── ChannelController
│   │   │       │   ├── CommentController
│   │   │       │   ├── EmailController
│   │   │       │   ├── PostController
│   │   │       │   ├── ReplyController
│   │   │       │   ├── SearchHistoryController
│   │   │       │   ├── ServerController
│   │   │       │   └── UserController
│   │   │       ├── entity/          # 数据库实体
│   │   │       │   ├── Broadcast
│   │   │       │   ├── BroadcastFavorite
│   │   │       │   ├── BroadcastHistory
│   │   │       │   ├── Channel
│   │   │       │   ├── Comment
│   │   │       │   ├── Post
│   │   │       │   ├── Reply
│   │   │       │   ├── SearchHistory
│   │   │       │   └── User
│   │   │       ├── dto/             # 数据传输对象
│   │   │       │   ├── CommentDTO
│   │   │       │   └── HashTag
│   │   │       ├── vo/              # 视图对象
│   │   │       │   ├── BroadcastItemVO
│   │   │       │   ├── BroadcastVO
│   │   │       │   ├── ChannelItemVO
│   │   │       │   ├── ChannelVO
│   │   │       │   ├── CommentVO
│   │   │       │   ├── PostVO
│   │   │       │   └── ReplyVO
│   │   │       ├── mapper/          # MyBatis映射器
│   │   │       │   ├── BroadcastMapper
│   │   │       │   ├── ChannelMapper
│   │   │       │   ├── CommentMapper
│   │   │       │   ├── PostMapper
│   │   │       │   ├── ReplyMapper
│   │   │       │   ├── SearchHistoryMapper
│   │   │       │   └── UserMapper
│   │   │       ├── result/          # 统一返回结果
│   │   │       │   ├── Result
│   │   │       │   └── ResultCodeEnum
│   │   │       ├── service/         # 服务层
│   │   │       │   └── impl/        # 服务实现
│   │   │       │       ├── BroadcastService
│   │   │       │       └── ChannelService
```

```
│   │   │       │       ├── ChannelService
│   │   │       │       ├── CommentService
│   │   │       │       ├── PostService
│   │   │       │       ├── ReplyService
│   │   │       │       ├── SearchHistoryService
│   │   │       │       └── UserService
│   │   │       └── utils/           # 工具类
│   │   │           ├── EmailUtil
│   │   │           ├── FileUtils
│   │   │           ├── TokenUtil
│   │   │           ├── VCodeUtil
│   │   │           └── IFmServerApplication
│   │   └── resources/
│   │       ├── static/              # 静态资源
│   │       │   └── images/
│   │       │       ├── channel/    # 频道图片
│   │       │       │   └── D.png
│   │       │       └── user.head/  # 用户头像
│   │       │           ├── 6_1732262627.png
│   │       │           ├── 7_1732262834.png
│   │       │           └── D.png
│   │       ├── application.properties # 应用配置
│   │       └── mybatis-config.xml   # MyBatis配置
│   └── test/                        # 测试代码
├── target/                          # 构建输出
│   ├── .gitattributes
│   ├── .gitignore
│   ├── mvnw                         # Maven包装器脚本
│   ├── mvnw.cmd
│   └── pom.xml                      # Maven项目配置
└── pom.xml                          # 主POM文件
```

- 核心模块:
  - 用户模块(登录注册)
  - 播客模块(上传、展示、播放)
  - 评论模块(添加、查询)

2.2.3 数据库结构

- 用户表(user):保存用户信息、权限、状态;
- 播客表(podcast):保存播客基本信息;
- 播放记录表(history):记录用户播放行为;
- 收藏/点赞/评论表(optional):用于记录用户互动行为。
- 关系:一对多(用户与评论)、多对多(用户与收藏)

1.5 项目计划与团队分工

见附录C:开发阶段进度计划与执行

二、需求分析与设计 [张皖男]

2.1 用户画像与场景分析

详细分析目标用户群体(学生、教师、管理员),以及具体的使用场景。

2.2 界面原型设计

2.2.1 交互设计原则

1.产品核心流程图

image-20241227171806144

2.功能摘要

image-20241227171849300

详细见附录D:产品界面详细说明

2.2.2 用户体验设计

- 设计清晰、直观的导航结构，保证用户能够快速找到所需功能；
- 操作流程简洁，减少用户完成任务的步骤和时间；

说明移动端交互设计的特殊考虑，如手势操作、屏幕适配、响应式设计等。

2.2.2 用户体验设计

描述如何优化移动端用户体验，包括加载速度、操作便捷性、视觉设计等。

三、系统架构设计 [李四]

3.1 整体架构设计

描述系统的整体架构，包括前端、后端、API、数据库、缓存、消息队列等各个部分的大致结构，可用示意图来进行描述，并详细说明每部分的具体作用，以及为什么这么设计，其优缺点是什么。

3.2 技术架构分层

3.2.1 表现层（前端）

- **模块组成**
  包含用户管理模块（登录注册）、内容浏览模块（首页、频道）、播放模块、创作中心和个人中心。

- **模块关系**
  各模块通过路由统一管理，支持模块间跳转和数据共享。

- **设计原因**
  分模块设计便于功能解耦和单独维护，提升代码复用率。

- **优缺点**
  优点：灵活、易扩展。
  缺点：初期开发成本较高，模块间接口需明确规范。

3.2.2 业务逻辑层（后端）

- **模块划分**
  用户服务、内容服务、权限服务、通知服务、统计服务等。

- **业务流程设计**
  请求由控制器接收，调用对应服务层处理业务逻辑，最终通过数据访问层操作数据库。

- **设计优势**
  层次分明，易于单元测试和维护。

3.2.3 数据访问层

- **数据**库设计
  采用关系型数据库，设计规范的ER模型，确保数据完整性和一致性。

3.3 微服务架构设计（如适用）

无

3.4 移动端特色架构考虑。

无

四、API设计 [张皖男、徐显舜]

4.1 API设计原则

在本项目中，我们采用了RESTful API设计原则，以保证接口的规范性、易用性和可扩展性，具体包括：

1. 资源导向设计

- API以资源为核心，使用名词表示资源（如`/users`、`/courses`）
- 每个URL代表唯一资源，便于理解和管理。

2. 版本控制

- 通过URL路径版本号管理接口版本，如 `/api/v1/users`。
- 保持向后兼容，支持平滑升级。

4.2 接口文档

见附录A: 详细API文档

4.4 接口测试

测试策略：结合单元测试和集成测试，覆盖所有关键接口。- **测试用例设计**：- 正常场景测试：验证接口基本功能正确。- 异常场景测试：模拟错误输入、权限不足、网络异常等。- 边界条件测试：测试参数最大/最小值、空值等。- **自动化测试**：- 使用Postman Collection和Newman实现自动化接口测试。- 集成CI/CD流程，实现每次代码提交自动触发接口测试。- 定期回归测试，确保接口变更不会破坏现有功能。

五、数据库设计 [张皖男]

5.1 数据模型设计

tohear项目中主要涉及用户、播客内容、订阅、评论、播放记录等数据。系统需要持久化的主要数据包括：

- **用户信息**：用户账号、昵称、头像、密码等基本信息。
- **播客节目**：包含节目标题、简介、作者信息、音频文件路径等。
- **订阅关系**：用户与播客的订阅状态。

- 评论信息：用户对节目的评论内容及时间。
- 播放记录：用户的播放历史和进度，支持续播功能。
- 缓存数据：音频缓存、本地离线数据等。

数据库采用 MySQL，理由如下：

- 关系型数据结构清晰，方便管理复杂的多表关系；
- 支持事务保证数据一致性；
- 社区广泛，工具链成熟；
- 与 SpringBoot 框架无缝集成，ORM支持良好。

5.2 数据库架构

5.2.1 核心数据表设计

表名	说明	主要字段示例
users	用户信息表	user_id, username, password, avatar
podcasts	播客节目表	podcast_id, title, description, author_id, audio_url
subscriptions	用户订阅关系	subscription_id, user_id, podcast_id, subscribe_time
comments	节目评论表	comment_id, user_id, podcast_id, content, create_time
play_records	播放记录表	record_id, user_id, podcast_id, last_position, update_time

表设计重点考虑：

- 主键统一使用自增ID，便于索引优化；
- 外键关联保证数据完整性，如subscriptions.user_id关联users.user_id；
- 支持多对多关系，如用户和播客的订阅。

5.2.2 索引设计

主键索引：所有表均设主键索引。

外键字段索引：如subscriptions.user_id和subscriptions.podcast_id设联合索引，优化订阅查询。

常用查询字段索引：如comments.podcast_id用于快速查询评论列表。

播放记录表user_id和podcast_id建立复合索引，加快播放记录的检索。

5.2.3 数据库性能优化

使用合理的索引减少查询时间；

对频繁更新的表进行合理分区或分表（如播放记录）；

采用连接池技术减少连接开销；

视情况使用缓存（Redis等）存储热点数据，如用户订阅列表；

定期清理过期数据，保持表的健康状态。

5.3 数据访问层实现

后端采用 Spring Data JPA 作为ORM框架，实现实体类与数据库表的映射；

通过接口定义实现数据访问层（DAO），利用Spring Boot自动配置简化开发；

实现分页查询、条件查询等复杂操作，保证系统响应速度；

使用事务管理确保数据操作的原子性和一致性。

5.4 数据同步与备份

移动端数据同步采用**增量同步机制**，保证用户数据在云端和本地一致；

通过定时任务同步服务器端播放记录和订阅状态；

后端采用定期备份数据库的方式，确保数据安全；

设计备份恢复方案，包括全量备份和增量备份，支持快速恢复；

采用主从复制提升数据库可用性和容灾能力。

六、前端实现 [徐显舜、张皖男]

6.1 技术栈与开发环境

本项目前端采用 **uni-app** 框架，支持多端（微信小程序、Android、iOS、Web）统一开发，提升开发效率。
开发工具主要使用 **HBuilderX**，支持一键调试与打包。构建流程包括代码编写、模拟器预览、真机调试及最终发布。
技术栈包括：Vue.js、uni-app组件库、Vuex状态管理等。

项目结构划分：

```
tohear_clent/
├── vite/                    # Vite构建工具
├── node_modules/            # 项目依赖
└── src/                     # 源代码目录
    ├── assets/
    │   └── iconfont.css     # 图标字体
    ├── components/          # 公共组件
    │   ├── broadcast-item/  # 广播组件
    │   ├── channel-item/    # 频道组件
    │   ├── comment-item/    # 评论组件
    │   ├── drop-down/       # 下拉组件
    │   ├── edit-wind/       # 编辑窗口
    │   ├── head-generic/    # 通用头部
    │   ├── my-button/       # 自定义按钮
    │   ├── my-head/         # 个人中心头部
    │   ├── player-bar/      # 播放器控制条
    │   ├── popupwindow/     # 弹窗组件
    │   ├── post-head/       # 帖子头部
    │   ├── post-interaction/# 帖子交互
    │   ├── post-item/       # 帖子项
    │   ├── reply-item/      # 回复项
    │   ├── search-box/      # 搜索框
    │   ├── sort-method-box/ # 排序选择
    │   ├── tabbar/          # 底部导航
    │   ├── tags-tabs/       # 标签页
    │   └── useritem/        # 用户项
    ├── pages/               # 页面目录
    │   ├── broadcast/       # 广播页
    │   ├── channel/         # 频道页
    │   ├── community/       # 社区页
    │   ├── creation/        # 创作页
    │   ├── index/           # 首页
    │   ├── login-register/  # 登录注册
    │   ├── my/              # 个人中心
    │   ├── player/          # 播放器
    │   ├── search-pages/    # 搜索页
    │   ├── test/            # 测试页
    │   └── user/            # 用户页
    ├── request/             # 请求封装
    │   ├── apis.ts          # API接口
    │   └── index.ts         # 请求核心
    ├── static/              # 静态资源
    │   └── tab_icons/       # 标签图标
    │       ├── bg.png       # 背景图
    │       └── logo.png     # Logo
    ├── stores/              # 状态管理
    │   ├── base.ts          # 基础Store
    │   ├── counter.ts       # 计数器Store
    │   ├── player-副本.ts   # 播放器备份
    │   ├── player.ts        # 播放器状态
    │   └── user.ts          # 用户状态
    ├── styles/              # 样式文件
    │   └── tailwind.css     # Tailwind样式
    ├── uni_modules/         # uni-app模块
    ├── utils/               # 工具函数
    │   ├── broadcastSort.ts # 广播排序
    │   ├── channelSort.ts   # 频道排序
    │   ├── checkPassword.ts # 密码验证
    │   ├── commentSort.ts   # 评论排序
    │   ├── durationFormatting.ts # 时长格式化
    │   ├── initPlayer.ts    # 播放器初始化
    │   ├── logOut.ts        # 登出逻辑
    │   ├── numConversion.js # 数字转换
    │   ├── postSort.ts      # 帖子排序
    │   └── timeChange.ts    # 时间转换
    ├── App.vue              # 根组件
    ├── env.d.ts             # 环境类型
    ├── main.ts              # 主入口
    ├── manifest.json        # 应用配置
    ├── pages.json           # 路由配置
    ├── shime-uni.d.ts       # uni类型声明
    ├── uni.scss             # 全局样式
    ├── .gitignore           # Git配置
    └── index.html           # HTML入口
```

6.2 核心功能模块实现

6.2.1 用户管理模块

功能涵盖登录、注册、个人信息查看与修改。
- 使用表单验证确保输入合法性，密码采用前端简单加密后传输。
- 登录时通过调用后端API验证用户身份，返回token进行后续鉴权。
- 优点：接口清晰，前后端分离，用户体验良好；缺点：依赖网络，首次加载可能稍慢。
- 实现难点：异步请求的状态管理和错误处理。

6.2.3 数据展示模块

微信图片_20250611060235

1. 采用的核心技术

(1) 前端框架：Vue.js + 自定义组件库

- 技术栈
  - 基于 **Vue 3**（`<template>`语法）
  - 使用类似 **uView UI** 的自定义组件库（`uv-text`、`uv-avatar`、`uv-image`等）
  - 状态管理：Vuex 或 Pinia（图中未展示，但需管理`channelDetail`数据）

(2) 关键实现特性

技术点	说明
组件化开发	通过`uv-*`系列组件封装通用功能（如头像、文本、图片懒加载）
动态数据绑定	`:text="channelDetail.channelTitle"` 实现数据驱动视图更新
响应式布局	`width="100%"` + `mode="aspectFill"` 适配不同屏幕尺寸
性能优化	图片懒加载（`lazy-load="true"`）减少首屏加载压力

2. 优缺点分析

优点

优势	说明
高复用性	自定义组件（如`uv-text`）可在全项目复用，减少重复代码
维护便捷	模块化结构清晰，数据与视图分离，便于迭代
性能优化	懒加载、样式隔离（`customStyle`）提升页面渲染效率
开发效率高	组件库提供现成功能（如多行文本截断`lines="3"`）

缺点

劣势	说明
框架强依赖	深度绑定Vue生态，迁移或替换成本高
学习成本	需掌握自定义组件库的API（如`uv-image`的`mode`属性）
灵活性受限	组件库功能固定，特殊需求需二次开发（如自定义懒加载逻辑）

6.3 性能优化

采用代码分包和按需加载减少首屏加载时间。
- 优化内存管理，避免内存泄漏。

- 使用请求合并、缓存和重试机制，提升网络请求效率和稳定性。
- 通过调试工具分析性能瓶颈，针对性优化。

6.4 兼容性处理

- **多设备适配**
  使用响应式布局和弹性设计，保证页面在手机、平板、PC等多种设备上均能良好显示。通过 `flexbox` 和 `media queries` 动态调整布局。
  - **跨平台支持**
    利用 uni-app 框架的多端编译能力，统一管理代码，iOS、H5等多个平台，避免重复开发。
  - **浏览器兼容性**
    采用现代Web标准，兼容主流浏览器（Chrome、Safari、Firefox、Edge等）。对不支持的API使用 polyfill 进行兼容处理。
  - **版本兼容**
    针对不同系统版本（如 Android 7 及以上，iOS 11 及以上）测试功能表现，调整不兼容功能或提示用户更新版本。
  - **性能兼容**
    针对低性能设备优化资源加载，减少动画和复杂计算，保证流畅体验。
  - **异常处理**
    增加兼容性异常捕获机制，避免因设备差异导致的程序崩溃，提升稳定性。

七、后端实现 [徐显舜、张皖男]

7.1 技术栈与架构

后端采用Spring Boot框架，基于Java语言开发，理由包括：

- **成熟稳定**：Spring Boot生态完善，社区活跃，支持快速开发。
- **模块化**：易于分层设计，方便业务逻辑划分和维护。
- **丰富组件**：内置安全框架（Spring Security）、数据访问（Spring Data JPA）、缓存支持等。
- **易扩展性**：支持微服务架构，方便未来扩展。
- **技术栈**：Spring Boot + MyBatis Plus
- 项目结构：

```
tohear_server/
├── .idea/                    # IDEA配置文件
├── .mvn/                    # Maven包装器
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com.pxx.ifmserver/
│   │   │       ├── config/          # 配置类
│   │   │       │   ├── CorsConfig   # 跨域配置
│   │   │       │   └── WebConfig    # Web配置
│   │   │       ├── controller/      # 控制器层
│   │   │       │   ├── BroadcastController
│   │   │       │   ├── ChannelController
│   │   │       │   ├── CommentController
│   │   │       │   ├── EmailController
│   │   │       │   ├── PostController
│   │   │       │   ├── ReplyController
```

```
│   │   │   │   ├── SearchHistoryController
│   │   │   │   ├── ServerController
│   │   │   │   └── UserController
│   │   │   ├── entity/          # 数据库实体
│   │   │   │   ├── Broadcast
│   │   │   │   ├── BroadcastFavorite
│   │   │   │   ├── BroadcastHistory
│   │   │   │   ├── Channel
│   │   │   │   ├── Comment
│   │   │   │   ├── Post
│   │   │   │   ├── Reply
│   │   │   │   ├── SearchHistory
│   │   │   │   └── User
│   │   │   ├── dto/             # 数据传输对象
│   │   │   │   ├── CommentDTO
│   │   │   │   └── HashTag
│   │   │   ├── vo/              # 视图对象
│   │   │   │   ├── BroadcastItemVO
│   │   │   │   ├── BroadcastVO
│   │   │   │   ├── ChannelItemVO
│   │   │   │   ├── ChannelVO
│   │   │   │   ├── CommentVO
│   │   │   │   ├── PostVO
│   │   │   │   └── ReplyVO
│   │   │   ├── mapper/          # MyBatis映射器
│   │   │   │   ├── BroadcastMapper
│   │   │   │   ├── ChannelMapper
│   │   │   │   ├── CommentMapper
│   │   │   │   ├── PostMapper
│   │   │   │   ├── ReplyMapper
│   │   │   │   ├── SearchHistoryMapper
│   │   │   │   └── UserMapper
│   │   │   ├── result/          # 统一返回结果
│   │   │   │   ├── Result
│   │   │   │   └── ResultCodeEnum
│   │   │   ├── service/         # 服务层
│   │   │   │   └── impl/        # 服务实现
│   │   │   │       ├── BroadcastService
│   │   │   │       ├── ChannelService
│   │   │   │       ├── CommentService
│   │   │   │       ├── PostService
│   │   │   │       ├── ReplyService
│   │   │   │       ├── SearchHistoryService
│   │   │   │       └── UserService
│   │   │   └── utils/           # 工具类
│   │   │       ├── EmailUtil
│   │   │       ├── FileUtils
│   │   │       ├── TokenUtil
│   │   │       ├── VCodeUtil
│   │   │       └── IFmServerApplication
│   │   └── resources/
│   │       ├── static/              # 静态资源
│   │       │   └── images/
│   │       │       ├── channel/    # 频道图片
│   │       │       │   └── D.png
│   │       │       └── user.head/  # 用户头像
│   │       │           ├── 6_1732262627.png
│   │       │           ├── 7_1732262834.png
│   │       │           └── D.png
│   │       ├── application.properties # 应用配置
│   │       └── mybatis-config.xml   # MyBatis配置
│   └── test/                        # 测试代码
├── target/                          # 构建输出
│   ├── .gitattributes
│   ├── .gitignore
│   ├── mvnw                         # Maven包装器脚本
│   ├── mvnw.cmd
│   └── pom.xml                      # Maven项目配置
└── pom.xml                          # 主POM文件
```

数据库选用MySQL，配合Redis缓存提升性能。

7.2 核心业务模块实现

7.2.1 用户认证与授权

支持用户登录、注册

八、安全设计 [徐显舜]

8.1 安全威胁分析

数据泄露：数据库或通信过程中的敏感数据可能被非法获取。

8.2 安全防护措施

8.2.1 身份认证与授权

后端每次请求均验证 token 合法性和有效期。

8.2.2 数据加密

数据传输：前后端采用 HTTPS 协议进行通信，防止中间人监听。

8.3 隐私保护

设置隐私协议，用户使用前需明确授权同意。

九、系统测试 [张皖男]

- 单元测试（JUnit）：
  - 测试服务层、工具类逻辑正确性
- 接口测试：
  - 使用 Postman 逐一验证接口请求与返回数据
- 前端测试：
  - 手动测试主流程：登录 → 播客浏览 → 播放 → 评论/收藏
- 性能测试（可选）：
  - 使用 Apache JMeter 模拟并发用户访问
- Bug 跟踪工具推荐：语雀、飞书表格、TAPD 等

十、性能优化 [张皖男、徐显舜]

10.1 前端性能优化

10.1.1 页面加载优化

移除无效组件和冗余样式文件，减少初始体积。

10.1.2 资源优化

所有图标和 UI 元素使用矢量图(SVG)或 WebP 格式压缩图像资源。

10.2 后端性能优化

10.2.1 数据库优化

分多个表输入数据

13.3 网络优化

13.3.1 CDN应用

13.3.2 HTTP优化

十一、功能展示 [徐显舜]

11.1 系统演示

 演示视频链接(示例

11.2 用户体验测试

团队对系统进行了体验测试，测试方式为：

- 功能测试；

 反馈总结如下：

评价维度	用户反馈
界面美观度	简洁清晰，结构清晰，颜色搭配良好
操作流畅性	页面跳转速度快，响应及时
功能实用性	实用性强
使用难易度	功能入口清晰，易上手

14.1 性能测试结果

测试结论，系统性能表现稳定，接口响应时间在可接受范围内，服务器负载正常。

十二、项目管理与协作 [徐显舜]

12.1 开发流程

本项目采用了敏捷开发方法论，以小步快跑、快速迭代的方式推进开发进度。开发流程主要分为以下几个阶段：

1. 需求分析阶段：明确产品目标和核心功能，绘制 ER 图与原型图。
2. 任务拆解阶段：将项目划分为前端、后端和数据库三个部分，并细化为多个具体模块。
3. 迭代开发阶段：
    - 每周进行功能划分和任务分配；

    - 每日或每两日同步开发进度，进行问题讨论与解决；
    - 每个小模块完成后进行集成测试与验收。
4. **部署与调试阶段**：前后端联调，进行 bug 修复与功能补充，最终部署运行环境。

这种灵活的开发方式使得项目能够快速响应问题，提升团队协同效率。

12.2 协作工具

为保证高效协作与项目可控性，团队使用了多种协作工具：

- **版本控制工具**：**Git + GitHub Desktop**
    - 通过 Git 进行版本管理；
    - 每位成员负责不同模块，定期 push/pull 合并代码；
    - 采用分支策略（如 dev 分支、feature 分支）以避免冲突。

12.3 质量保证

为确保代码质量与系统稳定性，团队采用了以下质量控制措施：

- **代码 Review**：
    - 每个模块提交后，至少由另一名成员进行代码审查；
    - 审查内容包括代码规范、逻辑正确性、重复/冗余代码等。
- **功能测试**：
    - 各模块开发完成后，进行功能测试确保其符合需求；
    - 联调阶段进行回归测试，防止修改引入新 bug。
- **统一编码规范**：
    - 前端遵循 uni-app 的组件开发规范；
    - 后端遵循 Spring Boot 的标准结构与 RESTful API 风格；
    - 使用 Prettier / IDEA 格式化工具保持代码风格一致。

十三、成果与交付物 [张皖男]

13.1 项目交付清单

- 前端代码: wx_class项目
- 后端代码: class_backend项目
- 数据库脚本: database/目录
- API文档: docs/api/目录
- 原型设计文件: docs/design/目录
- 项目演示视频: docs/videos/目录
- 用户手册: docs/user_manual.md
- 部署文档: docs/deployment.md
- 测试报告: docs/test_report.md

十四、总结与展望 [张皖男、徐显舜]

14.1 项目总结

本项目基于 **uni-app + Spring Boot + MySQL** 构建，实现了一个听书类小程序的前后端分离架构。团队围绕用户登录、音频播放、用户信息管理、内容展示等核心功能进行了开发和优化。项目总体按计划完成，系统结构清晰，功能模块划分明确，具备良好的可扩展性与可维护性。

主要成果包括：

- 完成前端页面的搭建与基本交互逻辑实现；

- 搭建稳定的后端服务，支持基本的数据处理；
- 设计并实现 MySQL 数据库结构，完成数据存储与管理；
- 实现了基础的用户登录、内容展示、播放记录等核心功能。

14.2 技术收获

在项目开发过程中，团队成员在以下方面取得了显著提升：

- 熟悉并掌握了 **uni-app** 跨端开发框架，了解其组件化设计与页面跳转机制；
- 提升了对 **Spring Boot** 框架的理解，能够完成接口设计、参数校验和服务编写；
- 掌握了 **MySQL** 数据库建模、查询优化与数据交互 技能；
- 学习并实践了 **前后端分离开发流程**，包括接口联调、跨域处理等；
- 团队协作能力与项目管理能力显著提升，具备较强的问题解决能力。

14.3 问题与反思

项目过程中也遇到了一些挑战与问题，包括但不限于：

- **前后端接口不一致**：初期缺乏统一 API 文档，导致接口传参错误；
  - **解决方案**：进行接口文档管理，确保前后端对齐；
- **开发进度偏差**：部分模块开发时间超出预期；
  - **解决方案**：后期通过每日站会和任务分配机制进行进度追踪；
- **数据库设计初期不合理**：字段设计不规范，冗余较多；
  - **解决方案**：通过反复优化 ER 图，重构数据表结构，提升数据一致性与查询效率。

经验教训：**早期设计非常关键，应在开发前充分沟通，确定标准和接口规范。**

14.4 未来展望

当前项目仅实现了基础的音频内容管理与用户功能，后续可以进一步拓展：

- 引入 **AI 推荐算法**，实现个性化内容推送；
- 支持 **内容创作者入驻与上传音频功能**，构建平台生态；
- 实现 **后台管理系统**，提供数据统计、内容审核、用户管理等功能；
- 适配 **小程序平台发布**（如微信、支付宝、抖音）并进行用户测试。

14.5 商业化前景

随着内容消费持续增长，本项目具备一定的商业价值和市场潜力：

- **目标用户**：关注音频内容消费的年轻群体、通勤用户、泛知识类内容爱好者；
- **商业模式**：可探索广告变现、会员订阅、知识付费、内容打赏等；
- **差异化优势**：聚焦小众内容、强调个性推荐、提升用户体验；
- **合作方向**：可与音频内容创作者、MCN 机构、知识平台等开展合作；
- **技术壁垒**：前后端分离、跨端支持、音频处理能力为系统的核心优势。

参考文献

附录

附录A: 详细API文档

附录B: 数据库表结构