

上海電力大學

Python 大作业



题 目： 智能音乐推荐系统

学 号： 20221560

姓 名： 杨裕凯

院 系： 计算机科学与技术学院

专业年级： 2022 级计算机科学与技术(卓越培养计划)

2024 年 12 月 23 日

目录

第一章、 引言	1
第二章、 需求分析	1
2.1 项目背景与目标	1
2.2 系统需求	2
2.3 系统实现	2
2.4 技术需求	3
2.5 功能细化	3
2.6 预期效果	3
第三章、 系统设计	4
3.1 功能模块划分	4
3.2 系统数据库设计及实现	4
3.3 系统技术架构	8
第四章、 系统核心模块设计与实现	13
第五章、 系统总结	27
5.1 已实现功能	27
5.2 系统值得改进（没有实现，需要后面如何规划实现）	39
5.3 心得体会	40
参考文献	41

智能音乐推荐系统

第一章、引言

在数字化时代，音乐已经成为人们日常生活中不可或缺的精神食粮。随着音乐流媒体平台的普及，用户可以接触到海量的音乐资源，但同时也面临着选择困难的问题。如何在庞大的音乐库中找到符合个人品味的音乐，成为了一个亟待解决的问题。智能音乐推荐系统正是为了解决这一痛点而生。

在当前的音乐市场中，尽管各大平台都提供了音乐推荐服务，但许多推荐结果往往不够个性化，无法准确捕捉用户的音乐品味变化。此外，传统的音乐推荐方式往往过于依赖热门度和流行趋势，忽视了用户的个性化需求和长尾音乐的价值。因此，开发一个更智能、更个性化的音乐推荐系统具有重要意义。

本项目旨在创建一个集音乐播放、个人音乐管理和智能推荐于一体的综合性平台。通过该平台，用户不仅可以享受基础的音乐播放功能，还能够通过收藏、评论、打分等方式表达自己的音乐喜好，系统则基于这些用户行为数据，结合先进的推荐算法，为用户提供更加精准的音乐推荐服务。

本项目采用 Python 作为主要开发语言，后端使用 Flask 框架提供稳定高效的服务，前端则采用 Vue3 框架打造现代化的用户界面。这种技术架构既能确保系统的可靠性和性能，又能为用户提供流畅的使用体验。通过这个平台，我们期望能够帮助用户发现更多优质音乐，丰富他们的音乐生活，提升音乐欣赏体验。

第二章、需求分析

2.1 项目背景与目标

在数字化时代背景下，音乐流媒体服务已成为年轻人获取音乐的主要途径。本项目旨在开发一个智能音乐推荐系统，以培养学生在人工智能、数据分析和全栈开发领域的实践能力。该系统将整合现代推荐算法技术，为用户提供个性化的音乐服务体验。

在技术层面，本项目将让学生深入理解并运用 Python 编程语言，掌握 Flask 后端框架和 Vue3 前端框架的开发技能。通过实现音乐播放、用户行为分析、个性化推荐等功能，学生能够系统地学习和应用数据处理、算法设计、用户界面开发等多个领域的知识。

在应用层面，该系统将为用户提供一个智能化的音乐平台，通过收集和分析用户的音乐喜好、收听习惯和评价反馈，建立精准的用户画像，从而实现更有针对性的音乐推荐。这不仅能提升用户的音乐体验，也能帮助他们更好地发现和欣赏不同风格的音乐作品。

通过本项目的开发，学生将获得全面的软件工程实践经验，深入理解智能算法在日常生活中的实际应用，为未来的职业发展打下坚实基础。

2.2 系统需求

2.2.1 必选功能：

1. 音乐信息编辑
2. 播放、快进、暂停、停止、下一首、收藏、评论、打分等功能
3. 我的音乐喜好信息维护
4. 我的歌单维护
5. 推荐歌单（选用不同推荐算法）

2.2.2 技术要求：

- 编程语言：Python
- 后端框架：Flask
- 前端框架：Vue3

2.3 系统实现

1) 音乐信息管理功能：

- 支持音乐基本信息（歌名、歌手等）的编辑和管理
- 提供音乐标签分类系统
- 支持音乐文件的存储和管理
- 支持管理员音乐导入

2) 音乐播放核心功能：

- 实现基础播放控制：播放、暂停、快进、停止、下一首
- 支持音乐收藏、评论和打分功能
- 提供播放队列控制

3) 个人音乐喜好管理：

- 记录用户的音乐偏好
- 提供音乐收听历史记录和数据统计分析

4) 歌单功能实现：

- 支持创建和删除个人歌单
- 支持歌曲收藏到歌单功能

5) 智能推荐系统:

- 基于协同过滤算法的个性化音乐推荐
- 根据用户设置喜好的场景化推荐

此外，系统还将提供听歌习惯统计、听歌热榜等增值功能，以增强用户体验和平台的吸引力。

2.4 技术需求

- 1) 编程语言: Python, 以其简洁性和强大的库支持, 适合快速开发和原型制作。
- 2) 后端框架: Flask, 一个轻量级的 Web 应用框架, 适合构建简单的 RESTful APIs。
- 3) 前端框架: Vue3, 一个渐进式 JavaScript 框架, 用于构建用户界面, 提供响应式和动态的用户交互体验。

2.5 功能细化

- 1) 用户注册与登录
- 2) 智能音乐推荐: 使用协同过滤算法, 在主页自动生成智能推荐歌曲及歌曲、歌手排行榜信息。
- 3) 音乐列表: 实现所有歌曲的音乐列表展示, 同时支持搜索, 确保用户的查找体验
- 4) 音乐信息展示: 展示音乐的基本信息, 如音乐名、歌手、均分等。支持用户打分、评论功能。同时管理员还可以实现音乐基本信息的编辑。
- 5) 音乐播放: 设置音乐队列播放功能, 实现音乐播放、快进、暂停、停止、下一首功能, 确保用户的畅听体验。
- 6) 歌单创建: 开设了用户的创建、删除歌单功能, 同时支持用户收藏歌曲到指定歌单功能的实现。在歌单中, 可以将歌曲播放、添加到队列等。
- 7) 个人信息: 支持个人用户名、密码修改, 同时支持用户喜好信息维护。同时, 实现了用户的听歌习惯统计, 用户可以看到自己最近听取了什么歌曲, 最喜欢听的歌是哪一种类的歌曲, 以及听的最多的歌手。

2.6 预期效果

本智能音乐推荐系统将实现一个功能完整、界面友好的音乐平台。用户可以

便捷地进行音乐播放、歌单管理和个性化设置。系统通过分析用户的听歌习惯和评分数据,利用协同过滤算法为用户推荐符合其音乐品味的歌曲。在界面设计上,采用现代化的 Vue3 框架,确保操作流畅自然;在后端处理上,通过 Flask 框架提供稳定的数据服务和 API 支持。

第三章、系统设计

3.1 功能模块划分

1. 主页:

展示推荐音乐和热歌榜。

提供快速入口链接至音乐列表、歌单管理、个人信息管理等核心功能和实现搜索功能。

2. 用户注册登录模块:

允许新用户注册账户,收集基本信息并进行格式校验。

支持现有用户登录和个人信息管理。

3. 音乐列表模块:

展示所有音乐信息并提供完整的音乐播放控制功能,包括播放器基础功能和用户互动功能(收藏、评论、打分),同时支持管理员进行音乐信息编辑。

4. 歌单管理模块:

实现用户个人歌单的创建、编辑、删除等基础管理功能,支持音乐收藏到歌单并提供歌单内音乐的播放控制。

5. 个人信息管理模块:

提供用户基本信息修改和音乐喜好维护功能,展示个人听歌数据统计和历史记录,方便用户了解自己的音乐品味。

6. 搜索模块:

支持音乐搜索功能,实现快速精准的音乐查找体验。

3.2 系统数据库设计及实现

1. 逻辑设计

1) 用户表 users

- userid: VARCHAR(255), 主键, 用户 ID (登录用)。
- username: NVARCHAR(255), 用户名 (显示用, 可修改), 不允许为空。

- login_id: VARCHAR(255), 登录账号（不可修改），不允许为空，唯一。
- password: VARCHAR(255), 密码，不允许为空。
- user_identity: INT, 用户身份（0 普通用户，1 管理员），默认值 0。

2) 音乐表 music

- id: INT, 主键，自动增长。
- title: NVARCHAR(255), 歌曲名称，不允许为空。
- artist_name: NVARCHAR(255), 歌手名称，不允许为空。
- genre: NVARCHAR(255), 音乐类型，可以为空。
- play_count: INT, 播放次数，默认值 0。
- cover_url: NVARCHAR(255), 封面图片地址，默认值 '/static/music_img/default_cover.jpg'。

3) 歌单表 playlists

- id: INT, 主键，自动增长。
- userid: VARCHAR(255), 用户 ID，不允许为空。
- title: NVARCHAR(255), 歌单名称，不允许为空。
- description: NVARCHAR(MAX), 歌单描述，可以为空。
- created_at: DATETIME, 创建时间，默认当前时间。

4) 歌单-歌曲表 playlist_songs

- playlist_id: INT, 歌单 ID，主键的一部分。
- music_id: INT, 歌曲 ID，主键的一部分。
- 复合主键(playlist_id, music_id)。

5) 评论表 comments

- id: INT, 主键，自动增长。
- userid: VARCHAR(255), 用户 ID，不允许为空。
- music_id: INT, 歌曲 ID，不允许为空。
- comment_text: NVARCHAR(MAX), 评论内容，可以为空。
- created_at: DATETIME, 评论时间，默认当前时间。

6) 评分表 ratings

- id: INT, 主键，自动增长。

- userid: VARCHAR(255), 用户 ID, 不允许为空。
- music_id: INT, 歌曲 ID, 不允许为空。
- rating_value: INT, 评分值, 范围 1-5。

7) 播放历史表 user_play_history

- id: INT, 主键, 自动增长。
- userid: NVARCHAR(255), 用户 ID, 不允许为空。
- music_id: INT, 歌曲 ID, 不允许为空。
- played_at: DATETIME, 播放时间, 默认当前时间。

8) 用户偏好表 user_preferences

- id: INT, 主键, 自动增长。
- userid: VARCHAR(255), 用户 ID, 不允许为空。
- favorite_genres: NVARCHAR(255), 喜欢的音乐类型, 可以为空。
- favorite_artists: NVARCHAR(255), 喜欢的艺术家, 可以为空。
- listening_times: NVARCHAR(255), 常听时段, 可以为空。

2. 物理实现: SQL Server 建表

```
CREATE TABLE users (
    userid VARCHAR(255) PRIMARY KEY,           -- 用户ID (登录用)
    username NVARCHAR(255) NOT NULL,           -- 用户名 (显示用, 可修改)
    login_id VARCHAR(255) NOT NULL UNIQUE,      -- 登录账号 (不可修改)
    password VARCHAR(255) NOT NULL,            -- 密码
    user_identity INT DEFAULT 0                 -- 用户身份 (0普通用户, 1管理员)
);
```

图 1 建立用户表

```
CREATE TABLE music (
    id INT IDENTITY(1,1) PRIMARY KEY,          -- 自动增长的主键
    title nVARCHAR(255) NOT NULL,              -- 歌曲名称
    artist_name nVARCHAR(255) NOT NULL,        -- 歌手名称
    genre nVARCHAR(255),                       -- 音乐类型 (可以存储多个类型)
    play_count INT DEFAULT 0,                  -- 播放次数, 默认值为0
    cover_url nVARCHAR(255) DEFAULT '/static/music_img/default_cover.jpg' -- 专辑封面默认地址
);
```

图 2 创建音乐表


```

CREATE TABLE playlists (
    id          INT IDENTITY(1,1) PRIMARY KEY, -- 自动增长的主键
    userid      varchar(255) NOT NULL,         -- 用户ID
    title       nvarchar(255) NOT NULL,        -- 歌单名称
    description  nvarchar(max),                 -- 歌单描述
    created_at  DATETIME DEFAULT GETDATE(),    -- 创建时间，默认当前时间
);

```

图 3 创建歌单表

```

CREATE TABLE playlist_songs (
    playlist_id INT NOT NULL,                 -- 歌单ID
    music_id    INT NOT NULL,                 -- 歌曲ID
    PRIMARY KEY (playlist_id, music_id)      -- 复合主键
);

```

图 4 创建歌单-歌曲表

```

CREATE TABLE comments (
    id          INT IDENTITY(1,1) PRIMARY KEY, -- 自动增长的主键
    userid      varchar(255) NOT NULL,         -- 用户ID
    music_id    INT NOT NULL,                 -- 歌曲ID
    comment_text nvarchar(max),               -- 评论内容
    created_at  DATETIME DEFAULT GETDATE(),    -- 评论时间，默认当前时间
);

```

图 5 创建评论表

```

CREATE TABLE ratings (
    id INT IDENTITY(1,1) PRIMARY KEY, -- 自动增长的主键
    userid varchar(255) NOT NULL,      -- 用户ID
    music_id INT NOT NULL,             -- 歌曲ID
    rating_value INT CHECK (rating_value BETWEEN 1 AND 5), -- 评分值，范围为1到5
);

```

图 6 创建评分表

```
CREATE TABLE user_play_history (
    id INT IDENTITY(1,1) PRIMARY KEY,    -- 自动增长的主键
    userid nvarchar(255) NOT NULL,        -- 用户ID
    music_id INT NOT NULL,                -- 歌曲ID
    played_at DATETIME DEFAULT GETDATE(), -- 播放时间，默认当前时间
);
```

图 7 创建播放历史表

```
CREATE TABLE user_preferences
(
    id INT IDENTITY (1,1) PRIMARY KEY,
    userid VARCHAR(255) NOT NULL,
    favorite_genres NVARCHAR(255), -- 喜欢的音乐类型（用逗号分隔的字符串）
    favorite_artists NVARCHAR(255), -- 喜欢的艺术家（用逗号分隔的字符串）
    listening_times NVARCHAR(255) -- 常听时段（用逗号分隔的字符串）
)
```

图 8 用户偏好表

3.3 系统技术架构

3.3.1 前后端分离

- 1) 前后端分离：前端使用 Vue3 构建，后端使用 Flask，通过 RESTful API 进行通信。
- 2) 数据库交互：后端 Flask 应用通过 SQLAlchemy 与数据库交互，处理业务逻辑。
- 3) 状态管理：前端使用 sessionStorage 进行状态管理，保持组件间的状态同步。
- 4) 路由管理：Vue Router 负责前端页面的路由管理，实现单页面应用的导航。
- 5) 模块化开发：前端采用组件化开发，提高代码复用性和维护性。
- 6) 可扩展性：后端的 RESTful API 设计保证了系统的可扩展性，方便前后端分离开发和维护。

3.3.2 Vue3 前端

1) 项目结构

LoginPage.vue 、 HomePage.vue 、 MyPage.vue 、 RegisterPage.vue 、 SearchResult.vue 、 MyPlaylists.vue 、 PlaylistDetail.vue 、 MusicDetail.vue 、 MusicEdit.vue、MusicList.vue、MusicPlayer.vue 等：Vue 组件文件，用于构建前端页面。

index.js: 定义前端路由的配置文件，使用 Vue Router 管理页面路由。

app.vue: Vue 应用的入口文件，用于初始化 Vue 实例和配置。

/public/static、 components 等: 存放资源文件、组件、页面等的目录。

2) 技术选型

Vue.js: 渐进式 JavaScript 框架，用于构建用户界面。

Vue Router: 官方路由管理器，用于构建单页面应用。

3) 路由配置

根据 router/index.js 的内容，应用包含以下页面路由：

- a. 登录页面 (/login)
- b. 注册页面 (/register)
- c. 首页 (/)
- d. 音乐列表页面 (/music-list)
- e. 音乐详情页面 (/music/:id)
- f. 我的歌单页面 (/my-playlists)
- g. 歌单详情页面 (/playlist/:id)
- h. 音乐编辑页面 (/music/:id/edit)
- i. 个人页面 (/my)
- j. 搜索结果页面 (/search)

其中一些路由还设置了特殊的访问权限：首页需要用户登录才能访问（requiresAuth: true）；音乐编辑页面需要管理员权限（requiresAdmin: true）和登录状态（requiresAuth: true）

这些路由构成了一个完整的音乐平台的基本功能结构，包括用户认证、音乐管理、歌单管理、个人中心和搜索功能等模块。

4) 静态资源管理

public 目录：存放不经过 Vue 构建流程的静态资源，包括歌曲文件和封面文件。

5) 依赖管理

使用 npm 管理前端依赖，存放在 node_modules 目录。

6) 运行环境

- node.js v22.11.0
- Vue3
- @vue/cli 5.0.8

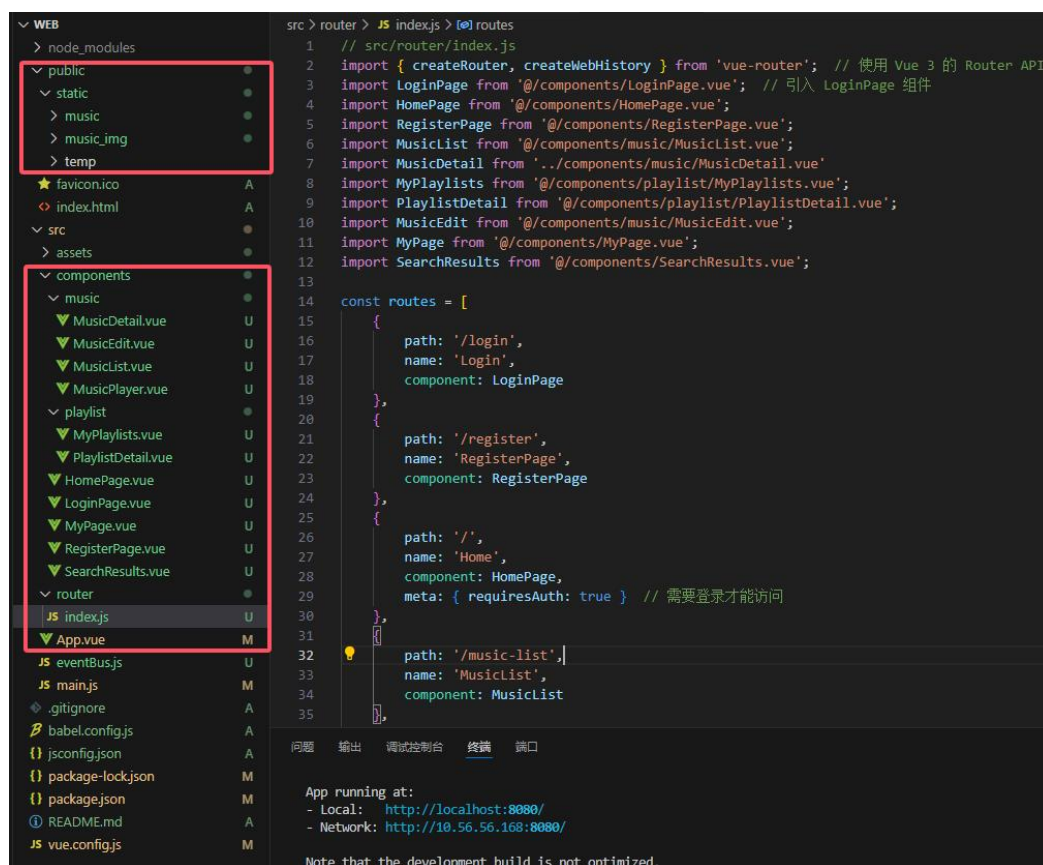


图 9 Vue3 前端项目文件结构

3.3.3 Flask 后端

1) 项目结构

- app.py: Flask 应用的主文件，通常包含应用创建。
- config.py: Flask 应用数据库的配置文件，包含使用的数据库的 url 以及账号密码等。

- `app._init_.py`: 项目的初始化文件，负责创建 Flask 应用实例、配置数据库连接、注册蓝图路由，是整个应用的入口点。
- `app.models.py`: 定义了项目所需的数据库表结构，包括用户、音乐、歌单、评论、评分等数据模型，使用 SQLAlchemy ORM 进行管理。
- `routes.auth.py`: 处理用户认证相关的路由和逻辑，包括用户注册、登录、修改用户名和密码等功能的实现。
- `routes.music.py`: 处理音乐相关的路由和逻辑，包括获取音乐列表、音乐详情、评论、评分、播放记录、音乐封面上传、音乐导入等功能的实现。
- `routes.playlist.py`: 处理歌单相关的路由和逻辑，包括创建歌单、获取歌单详情、添加/删除歌曲到歌单等功能的实现。
- `routes.rankings.py`: 处理音乐排行榜相关的路由和逻辑，包括获取歌手排行榜和热门歌曲排行榜的功能实现。
- `routes.recommendations.py`: 实现音乐推荐系统的核心逻辑，通过协同过滤算法分析用户的评分和播放历史，为用户生成个性化的音乐推荐列表。
- `routes.search.py`: 处理音乐搜索功能，支持通过歌名和歌手名进行模糊查询，返回匹配的音乐列表。
- `routes.user_preferences.py`: 处理用户偏好设置相关的路由和逻辑，包括保存和获取用户喜欢的音乐类型、艺术家和听歌时段等个性化设置。
- `routes.user_stats.py`: 处理用户统计数据相关的路由和逻辑，包括获取用户最常听的音乐类型、艺术家和最近播放记录等统计信息。
- `routes.static`: 包含对一些静态文件的预处理，包括将歌曲信息插入数据库当中。

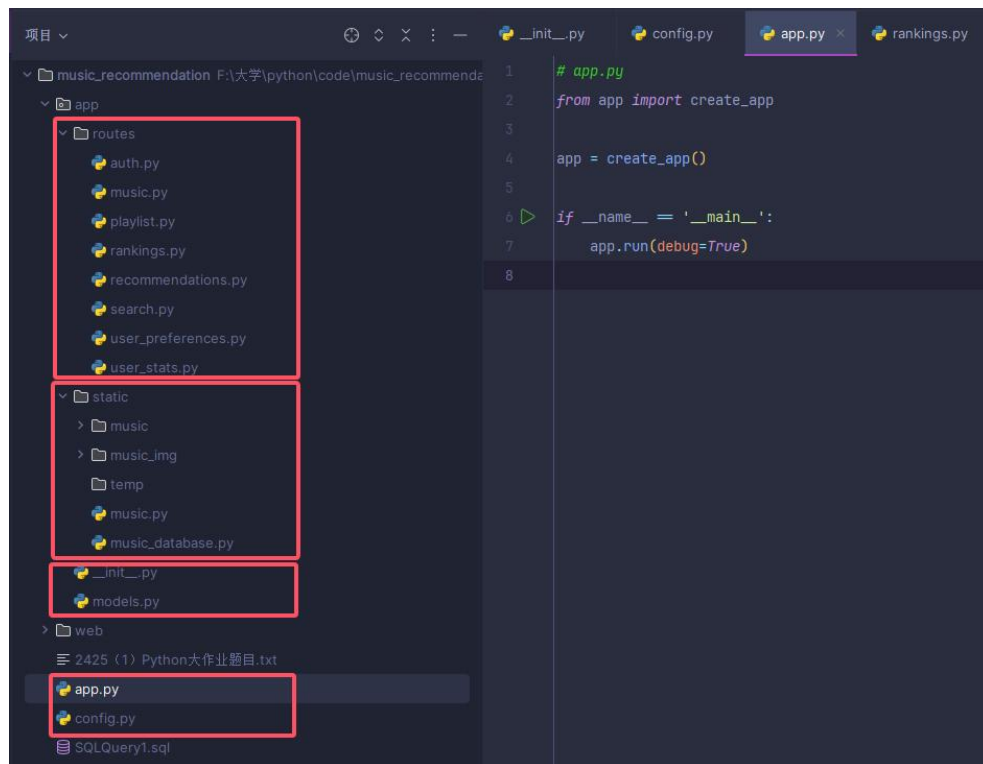


图 10 Flask 项目文件结构

2) 技术选型

- Flask: 轻量级的 Web 应用框架，适合快速开发。
- SQLAlchemy: 可选的 ORM 工具，用于数据库操作。

3) 依赖管理

使用 Anaconda 管理项目依赖。

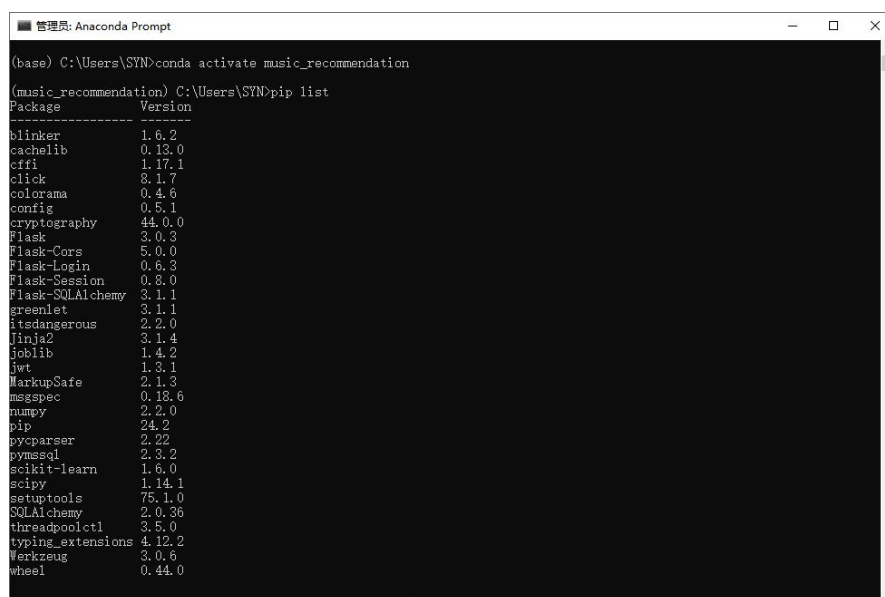


图 11 pip list

4) 运行环境

- Pycharm2023.3.2
- Python3.12.7

第四章、系统核心模块设计与实现

1. Flask 作为后端、Vue 作为前端的实现:

1) 前端 (Vue): 主要是 App.vue 和 index.js

(1) App.vue

① 模板 (template):

定义一个导航栏 nav (同时检测是否是登录界面, 如果是登录界面则不会出现导航栏), 里面有四个 router-link 指向其他的界面, 同时还有一个 input 来输入音乐搜索信息。

<router-view> 用于动态加载当前路由对应的组件。

MusicPlayer, 即音乐播放队列的前端展示界面。



```
<template>
  <div id="app">
    <div id="body">
      <!-- 导航栏 -->
      <nav v-if="!isLoginPage">
        <router-link to="/">音乐馆</router-link> |
        <router-link to="/music-list">音乐列表</router-link> |
        <router-link to="/my-playlists">我的歌单</router-link> |
        <input type="text" v-model="searchQuery" placeholder="搜索音乐" @keyup.enter="handleSearch" />
        <router-link to="/my">我的</router-link>
      </nav>

      <div class="main-content">
        <!-- 动态加载路由内容 -->
        <transition name="fade">
          <router-view /> <!-- 根据路由渲染相应的组件 -->
        </transition>
      </div>
      <MusicPlayer v-if="!isLoginPage && isLoggedIn" ref="player" />
    </div>
  </div>
</template>
```

图 12 App.vue 中<template>部分

② 脚本 (script):

根据'vue-router'中的 useRoute()获取当前路由对象, 通过 path 属性来获得当

前路径，从而判断是否在登录和注册界面。

根据 sessionStorage 来是否有用户数据来判断用户是否已经成功登录。

通过'vue-router'中的 useRouter()向指定的 path 发送请求从而实现搜索功能。

```
1 <script>
2 import { computed, ref } from 'vue';
3 import { useRoute, useRouter } from 'vue-router';
4 import MusicPlayer from './components/music/MusicPlayer.vue';
5
6 export default {
7   components: {
8     MusicPlayer // 注册MusicPlayer组件
9   },
10  setup() { // Vue3的组合式API入口函数
11    const route = useRoute(); // 获取当前路由对象
12    const router = useRouter(); // 获取路由器实例
13
14    // 计算属性: 判断是否为登录/注册页面
15    const isLoginPage = computed(() => {
16      return route.path === '/login' || route.path === '/register';
17    });
18
19    // 计算属性: 判断用户是否已登录
20    const isLoggedIn = computed(() => {
21      return !!sessionStorage.getItem('user');
22    });
23
24    const searchQuery = ref(''); // 创建响应式的搜索查询变量
25
26    // 处理搜索功能
27    const handleSearch = () => {
28      if (searchQuery.value.trim()) {
29        router.push({
30          path: '/search',
31          query: { q: searchQuery.value.trim() }
32        });
33      }
34    };
35
36    // 返回需要在模板中使用的数据和方法
37    return {
38      isLoginPage,
39      isLoggedIn,
40      searchQuery,
41      handleSearch
42    };
43  }
44 };
45 </script>
```

图 13 App.vue 中<script>部分

(2) router/index.js (script) :

使用'vue-router'定义应用的路由。每个页面路径映射到一个 Vue 组件，由 import 语句动态加载。路由配置包括首页、登录、注册、音乐列表、音乐详情、我的歌单、歌单详情、音乐编辑、个人信息、搜索结果等。使用 createRouter 和 createWebHistory 创建路由器，定义应用的路由表。

同时实现一个全局前置守卫，实现在未登录的时候将全部界面的请求重定向到登录界面。

```
// src/router/index.js
import { createRouter, createWebHistory } from 'vue-router'; // 使用 Vue 3 的 Router API
import LoginPage from '@components/LoginPage.vue'; // 引入 LoginPage 组件
import HomePage from '@components/HomePage.vue';
import RegisterPage from '@components/RegisterPage.vue';
import MusicList from '@components/music/MusicList.vue';
import MusicDetail from '../components/music/MusicDetail.vue'
import MyPlaylists from '@components/playlist/MyPlaylists.vue';
import PlaylistDetail from '@components/playlist/PlaylistDetail.vue';
import MusicEdit from '@components/music/MusicEdit.vue';
import MyPage from '@components/MyPage.vue';
import SearchResults from '@components/SearchResults.vue';
```

图 14 router.index.js 当中的 import

```
1 const routes = [
2   {path: '/login',name: 'Login',component: LoginPage},
3   {path: '/register',name: 'RegisterPage',component: RegisterPage},
4   // 需要登录才能够访问
5   {path: '/',name: 'Home',component: HomePage,meta: { requiresAuth: true }},
6   {path: '/music-list',name: 'MusicList',component: MusicList},
7   {path: '/music/:id',name: 'MusicDetail',component: MusicDetail},
8   {path: '/my-playlists',name: 'MyPlaylists',component: MyPlaylists},
9   {path: '/playlist/:id',name: 'PlaylistDetail',component: PlaylistDetail},
10  // 添加管理员权限要求
11  {path: '/music/:id/edit',name: 'MusicEdit',component: MusicEdit,meta: { requiresAuth: true,requiresAdmin: true}},
12  {path: '/my',name: 'MyPage',component: MyPage},
13  {path: '/search',name: 'Search',component: SearchResults}
14 ];
15
16 const router = createRouter({
17   history: createWebHistory(), // 使用 HTML5 History 模式
18   routes // 路由配置
19 });
```

图 15 router.index.js 当中的配置路由

```

// 全局前置守卫：检查是否需要登录
router.beforeEach( guard: (to : RouteLocationNormalized , from : RouteLocationNormalizedLoaded , next : NavigationGuardNext ) : void => {
    const user : string = sessionStorage.getItem( key: 'user' );
    let isLoggedIn : boolean = false;

    if (user) {
        try {
            const parsedUser = JSON.parse(user);
            isLoggedIn = parsedUser && parsedUser.userid && parsedUser.username;
        } catch (e) {
            console.error("Failed to parse user data in router:", e);
        }
    }

    // 如果目标路由需要登录且用户未登录
    if (to.meta.requiresAuth && !isLoggedIn) {
        // 重定向到登录页面
        next({ name: 'Login' });
    } else {
        // 允许访问页面
        next();
    }
});

```

图 16 router.index.js 当中的全局守卫

2) 后端 Flask (app.py)

(1) 初始化:

```

# app.py
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True)

```

图 17 创建一个 app 实例

```

# app/__init__.py
from flask import Flask
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy

from config import Config

# 创建一个数据库实例
db = SQLAlchemy()

2 个用法
🔍
def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)

    # 允许跨域请求
    CORS(app, supports_credentials=True) # 允许跨域请求时携带凭证

    # 初始化数据库
    db.init_app(app)

    # 注册蓝图 (模块化路由)
    from app.routes.auth import auth_bp
    from app.routes.music import music_bp
    from app.routes.playlist import playlist_bp
    from app.routes.user_preferences import user_pref_bp
    from app.routes.user_stats import user_stats_bp
    from app.routes.recommendations import recommendations_bp
    from app.routes.rankings import rankings_bp
    from app.routes.search import search_bp

    app.register_blueprint(auth_bp, url_prefix='/auth')
    app.register_blueprint(music_bp, url_prefix='/music')
    app.register_blueprint(playlist_bp, url_prefix='/playlist')
    app.register_blueprint(user_pref_bp, url_prefix='/preferences')
    app.register_blueprint(user_stats_bp, url_prefix='/user_stats')
    app.register_blueprint(recommendations_bp)
    app.register_blueprint(rankings_bp)
    app.register_blueprint(search_bp)

    return app

```

图 18 创建 app 函数

创建 Flask 应用实例并传入数据库的配置类。配置 CORS 允许跨域请求。
创建一个数据库示例并初始化。同时注册蓝图实现路由。

(2) 数据库操作:

① 完成配置文件 config.py:

A screenshot of a code editor showing the content of config.py. The code includes an import for os, a comment about 2 usages, and a Config class with two class attributes: SQLALCHEMY_DATABASE_URI and SQLALCHEMY_TRACK_MODIFICATIONS.

```
# config.py
import os

2 个用法
class Config:
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL',
                                             'mssql+pymssql://sa:yyk200412=localhost:1433/python_final')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

图 19 config.py

不难看出，文件当中已经定义好了连接数据库所需要使用的库 mssql+pymssql，以及连接所使用的账号（sa）、密码（yyk200412=）、url、连接的数据库（python_final）信息。

SQLALCHEMY_TRACK_MODIFICATIONS = False 的作用是关闭 SQLAlchemy 的对象修改跟踪系统，这可以减少内存使用并提高性能，因为不需要跟踪对象的变化。

② 完成数据库初始化:

A screenshot of a code editor showing the content of app/__init__.py. The code imports Flask, Flask-CORS, and Flask-SQLAlchemy, then imports the Config class from config. It then creates a SQLAlchemy database instance named db.

```
# app/__init__.py
from flask import Flask
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy

from config import Config

# 创建一个数据库实例
db = SQLAlchemy()
```

图 20 import 文件+创建一个数据库实例

A screenshot of a code editor showing the initialization of the database. It imports the db instance from the previous file and calls db.init_app(app) to initialize the database with the Flask application.

```
# 初始化数据库
db.init_app(app)
```

图 21 初始化数据库

③ 完成模块文件 models.py（以用户表为例）：



```
# app/models.py
from datetime import datetime, UTC

from sqlalchemy import PrimaryKeyConstraint, CheckConstraint, Unicode, UnicodeText

from app import db

8 个用法
class User(db.Model):
    __tablename__ = 'users'
    userid = db.Column(db.String(255), primary_key=True)
    username = db.Column(db.Unicode(255), nullable=False)
    login_id = db.Column(db.String(255), nullable=False, unique=True)
    password = db.Column(db.String(255), nullable=False)
    user_identity = db.Column(db.Integer, default=0)
```

图 22 User 配置

其中，__tablename__ 是 SQLAlchemy 自带的一个属性，用来填写数据库的表名。在需要时会自动连接数据库。同时，User 的各种变量的类型都与数据库相同，确保数据库查询、插入等操作的正确处理。

④ API 路由处理：定义了多个路由处理函数，如 register(), login(), change_username(), change_password() 等，它们与数据库交互，同时回复前端发送的请求。

⑤ 读取数据（以 user 表 login() 函数登录操作为例）：



```
# 查找用户
user = User.query.get(userid)
```

图 23 数据库查找函数

其中，userid 是从前端传输过来的数据。



```

# 查找用户

user = User.query.filter_by(login_id=login_id)

```

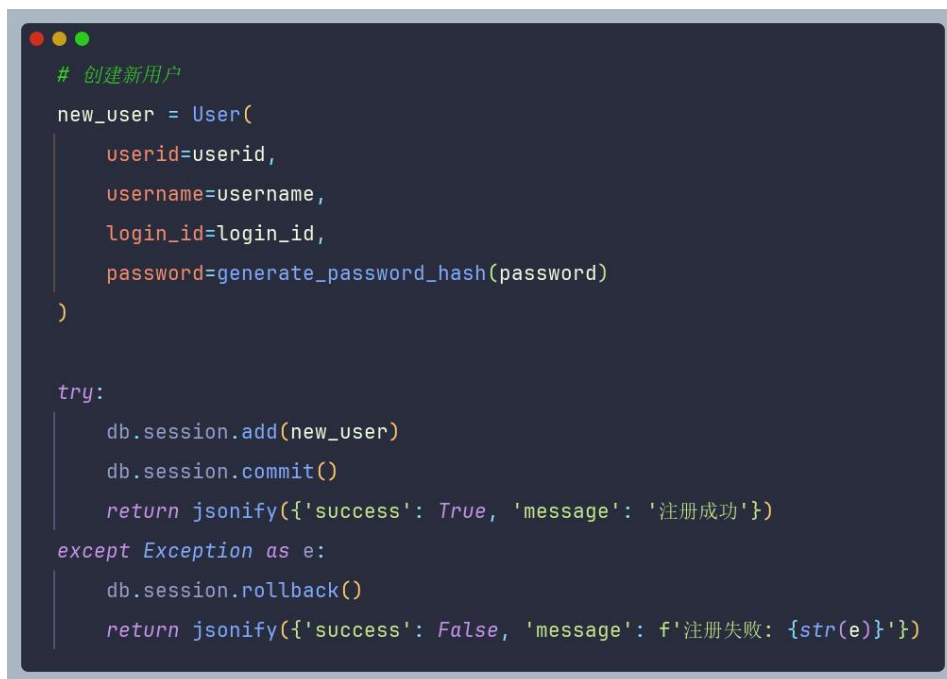
图 24 数据库查找函数

其中，login_id 是前端传输过来的数据。

注意，User.query.get(userid)是直接查询的主键，不可以使用其他值查询，返回的结果唯一。而 User.query.filter_by()是可以使用任意字段查询的，并且还可以实现查询的嵌套.filter_by().filter_by()，同时返回的结果不唯一，需要用.first()或者.all()来获取查询结果。

此时，user 就是一个实现了的 User 类。如果需要获得其他值，只需要使用 user.username 等语句即可获得查询到的用户名等信息。

⑥ 插入数据（以 user 表 register()注册为例）：



```

# 创建新用户

new_user = User(
    userid=userid,
    username=username,
    login_id=login_id,
    password=generate_password_hash(password)
)

try:
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'success': True, 'message': '注册成功'})
except Exception as e:
    db.session.rollback()
    return jsonify({'success': False, 'message': f'注册失败: {str(e)}'})

```

图 25 提交数据到数据库

可以看出提交数据的步骤就是创建一个新的 User 类，然后使用 db.session.add() 和 db.session.commit() 语句提交请求。如果出现错误则使用 db.session.rollback()来回溯。

⑦ 更新数据（以 user 表 change_username()更改用户名为例）：


```

# 查找用户
user = User.query.get(userid)
if not user:
    return jsonify({'success': False, 'message': '用户不存在'})

try:
    # 更新用户名
    user.username = new_username
    db.session.commit()
    return jsonify({
        'success': True,
        'message': '用户名修改成功',
        'user': {
            'userid': user.userid,
            'username': new_username,
            'user_identity': user.user_identity
        }
    })
except Exception as e:
    db.session.rollback()
    return jsonify({'success': False, 'message': f'用户名修改失败: {str(e)}'})

```

图 26 更新用户名

可以得到更新数据只需要在查找成功后更新类的变量，并实现提交语句即可完成数据库文件的更新。

⑧ 返回响应（以 user 表 register()注册为例）：

在图 26 当中有所展现，返回的是 json 化的数据。

(3) API 路由:

① 用户路由(auth.py):

```

app.register_blueprint(auth_bp, url_prefix='/auth')

```


图 27 注册蓝图

```

auth_bp = Blueprint(name='auth', __name__)

```

图 28 创建蓝图实例



```
@auth_bp.route(rule: '/register', methods=['POST'])
```

图 29 注解实现路由装饰器，用于定义 API 端点

后续路由装饰器不再展示。


`/auth/register methods=['POST']`: 处理新用户注册。在确认登录账号、用户名、密码全部输入、格式正确且没有登录账号重复之后，生成唯一的 `userid` 并创建新的用户。

`/auth/login methods=['POST']`: 处理用户登录。在确认登录账号、密码全部输入之后通过登录账号查找用户。查找到则验证密码是否正确。正确则返回用户信息。如果查找失败、密码错误则返回对应的错误信息。

`/auth/change-username methods=['POST']`: 处理用户更改用户名。确认用户名不为空之后根据 `userid` 查找用户并更新数据。


`/auth/change-password methods=['POST']`: 处理用户更改密码。确认输入数据合理之后查找用户并验证旧密码。确认旧密码正确之后更新密码。

② 音乐路由(music.py):



```
app.register_blueprint(music_bp, url_prefix='/music')
```

图 30 注册蓝图



```
music_bp = Blueprint(name: 'music', __name__)
```

图 31 创建蓝图实例

后续注册蓝图和创建蓝图实例不再展示。

`/music/list methods=['GET']`: 获取全部的音乐数据实现音乐列表功能。

`/music/<int:music_id> methods=['GET']`: 根据音乐的 `music_id` 获取具体的音乐信息。

`/music/<int:music_id>/comments methods=['GET']`: 根据音乐的 `music_id` 获取具体音乐的全部评论信息。

`/music/<int:music_id>/comments` `methods=['POST']`: 根据音乐的 `music_id` 插入用户的评论。

`/music/<int:music_id>/ratings` `methods=['GET']` : 获取指定歌曲的评分信息。自动实现获取该歌曲的平均分。

`/music/<int:music_id>/ratings` `methods=['POST']`: 保存用户对指定歌曲的评分信息。

`/music/<int:music_id>/play` `methods=['POST']`: 当用户播放歌曲之后, 实现指定歌曲的播放次数+1 且在 `user_play_history` 表中插入用户播放记录。

`/music/play-history/<string:userid>` `methods=['GET']`: 获取用户最近的 20 条播放记录。

`/music/upload-cover` `methods=['POST']`: 实现管理员更改音乐信息的封面时需先尝试将文件上传, 上传成功后才可以执行更改信息操作。

`/music/<int:music_id>` `methods=['PUT']`: 实现管理员更新指定音乐的信息, 包括封面路径等信息。

`/music/upload` `methods=['POST']`: 实现管理员上传音乐数据文件时先行保存音乐文件。保存成功才可以继续执行新插入音乐数据操作。

`/music/create` `methods=['POST']`: 实现管理员导入新音乐信息的操作。

③ 歌单路由(`playlist.py`):

`/playlist/user/<string:userid>` `methods=['GET']`: 获取指定用户的所有歌单的信息。

`/playlist/` `methods=['POST']`: 实现用户创建新歌单的操作。

`/playlist/<int:playlist_id>` `methods=['DELETE']`: 实现用户删除特定歌单操作。代码中实现先行删除表 `playlist_songs` 当中存储的歌单当中的音乐信息后再删除歌单。

`/playlist/<int:playlist_id>` `methods=['GET']`: 实现用户获取特定歌单整体信息。歌单整体信息包括歌单当中的歌曲及歌曲的信息, 使用 `join()` 连接表完成查询。

`/playlist/<int:playlist_id>/songs` `methods=['POST']`: 实现收藏歌曲功能。即将歌曲添加到指定的歌单当中。

/playlist/<int:playlist_id>/songs/<int:music_id> methods=['DELETE']: 实现用户从歌单当中删除歌曲。

④ 排名路由(rankings.py):

```
app.register_blueprint(rankings_bp)
```

图 32 注册蓝图

```
rankings_bp = Blueprint(name='rankings', __name__)
```

图 33 创建蓝图实例

因为 rankings_bp 在注册时没有设置 url_prefix, 所以直接访问 /rankings 即可。后续不再展示没有设置 url_prefix 的情况。

/rankings methods=['GET']: 通过将歌曲表和播放历史表连接来获得播放次数最多的歌曲和歌手名称。

```
# 歌手排行榜
artist_rankings = db.session.query(
    Music.artist_name,
    func.count(UserPlayHistory.id).label('play_count')
).join(
    UserPlayHistory, UserPlayHistory.music_id == Music.id
).group_by(
    Music.artist_name
).order_by(
    func.count(UserPlayHistory.id).desc()
).limit(10).all()

# 热歌排行榜
pop_index_rankings = db.session.query(
    Music.id,
    Music.title,
    Music.artist_name,
    Music.genre,
    func.count(UserPlayHistory.id).label('play_count')
).join(
    UserPlayHistory, UserPlayHistory.music_id == Music.id
).group_by(
    Music.id, Music.title, Music.artist_name, Music.genre
).order_by(
    func.count(UserPlayHistory.id).desc()
).limit(10).all()
```

图 34 连接获取歌手排行榜和热歌排行榜

⑤ 推荐路由(recommendations.py):

/recommendations methods=['GET']: 较为复杂, 实现了推荐功能。此功能使用了协同过滤算法。流程如下:

- 获取所有用户的评分数据
- 获取所有用户的播放历史数据, 并转化为用户在某个歌的播放次数
- 通过评分数据获取用户-歌曲矩阵
- 将播放次数转化为 1-5 分的分数 (和评分相同), 并补充用户-歌曲矩阵
- 计算用户相似度:
 - 输入两个用户的 id
 - 获取两个用户共同评价过 (或播放过) 的歌曲
 - 将两个用户对共同评价过 (或播放过) 的歌曲提取分数 (或播放次数) 组成向量
 - 求两个向量的余弦, 最终结果就是两个用户的余弦相似度
- 遍历用户, 获取和当前用户相似的用户的数据
- 将相似用户按相似度排序并取出前五名
- 将相似用户评分的歌曲取出, 并赋予分数 (用户排名*余弦相似度)
- 将所有推荐歌曲排序
- 如果推荐歌曲不足 15 首, 则从热歌榜中获取歌曲补足 15 首
- 获取所有推荐歌曲的具体信息
- 将推荐歌曲按照推荐顺序排序
- 返回推荐歌曲信息

⑥ 搜索路由(search.py):

/search methods=['GET']: 根据输入的关键词在数据库当中的 music 表当中进行歌曲名的模糊查询, 并返回查找到的歌曲信息。

⑦ 用户喜好路由(user_preferences.py):

/preferences/preferences methods=['POST']: 将用户输入的喜好信息保存到数据库当中。

/preferences/preferences methods=['GET']: 从数据库当中获取保存的用户的

喜好信息。

⑧ 用户状态路由(user_stats.py):

/user_stats/stats/music methods=['GET'] : 获取最近 30 天内的最常听的音乐类型统计、最常听的艺术家长统计和总播放次数。

/user_stats/stats/recent methods=['GET']: 获取用户获取最近播放的 10 首歌。

3) 实现原理及步骤

(1) 原理:

- ① 前端用户界面: 用户通过浏览器访问应用, Vue.js 框架渲染用户界面。
- ② 路由导航: 用户在应用内部导航, Vue Router 根据 URL 显示相应的页面。
- ③ 状态管理: sessionStorage 管理全局状态, 如用户认证信息, 确保应用状态的一致性。
- ④ API 通信: 前端通过 axios 请求与 Flask 后端的 RESTful API 进行通信。
- ⑤ 后端业务逻辑: Flask 处理 API 请求, 执行数据库查询、数据验证和其他业务逻辑。
- ⑥ 数据库交互: 使用 SQLAlchemy+mssql+pymssql 与 SQL Server 数据库进行交互, 执行 SQL 语句。
- ⑦ 响应处理: 后端将处理结果以 JSON 格式返回给前端。
- ⑧ 用户反馈: 前端根据后端响应更新 UI, 给用户反馈, 如登录成功、注册失败等。

(2) 步骤:

- ① 启动应用: 用户访问应用入口, 通常是 homepage.vue, 然后 App.vue 初始化。
- ② 页面渲染: Vue Router 根据路由配置渲染 App.vue 作为根组件。
- ③ 导航栏显示: 根据用户是否处于登录和注册界面, 显示不同的导航链接。
- ④ 用户操作: 用户进行操作, 如点击登录、注册或开始测试。
- ⑤ 发送请求: 前端组件通过 axios 发送请求到后端 API。
- ⑥ 后端处理: Flask 接收请求, 调用相应的处理函数, 如登录验证、数据插入等。
- ⑦ 数据库操作: 执行 SQL 查询或更新, 获取或修改数据。

- ⑧ 返回响应：处理完成后，Flask 将结果以 JSON 格式返回。
- ⑨ 前端更新：前端接收响应，更新页面或提示信息，如显示登录成功消息。
- ⑩ 用户认证：用户登录后，前端在本地存储保存认证信息，后端在每次请求中验证。
- ⑪ 应用关闭：用户关闭浏览器或导航至其他应用，前端清除活动状态，后端关闭数据库连接。

第五章、系统总结

5.1 已实现功能

1. 登录：



图 35 登录



图 36 未填写提示

The image shows a login interface with the title "登录" (Login). It contains two input fields: "登录账号" (Login Account) with the value "2" and "密码" (Password) with a single dot. Below the fields is a red error message "账号不存在" (Account does not exist). A blue "登录" (Login) button is present, and at the bottom, there is a link "还没有账户? 注册" (No account? Register).

图 37 登录账号错误提示

The image shows a login interface with the title "登录" (Login). It contains two input fields: "登录账号" (Login Account) with the value "20221560" and "密码" (Password) with a single dot. Below the fields is a red error message "密码错误" (Password error). A blue "登录" (Login) button is present, and at the bottom, there is a link "还没有账户? 注册" (No account? Register).

图 38 密码错误提示

2. 注册:

The image shows a registration interface with the title "注册" (Register). It contains four input fields: "登录账号" (Login Account) with the placeholder "请输入登录账号 (仅限英文字母和数字)", "用户名" (Username) with the placeholder "请输入用户名", "密码" (Password) with the placeholder "请输入密码 (至少6位)", and "确认密码" (Confirm Password) with the placeholder "请确认密码". A blue "注册" (Register) button is at the bottom, followed by a link "已有账户? 登录" (Already have an account? Login).

图 39 注册

注册

登录账号

123

用户名

oroite

密码

确认密码

两次输入的密码不一致

注册

已有账户? [登录](#)

图 40 密码不一致提示

localhost:8080/register

localhost:8080 显示
注册成功!

确定

注册

登录账号

yyk20221560

用户名

oroite

密码

确认密码

注册

已有账户? [登录](#)

图 41 注册成功提示

3. 登录后跳转到首页，包括推荐歌曲和排行榜信息。点击歌曲或者排行榜当中的歌曲，可以跳转到歌曲详情界面：

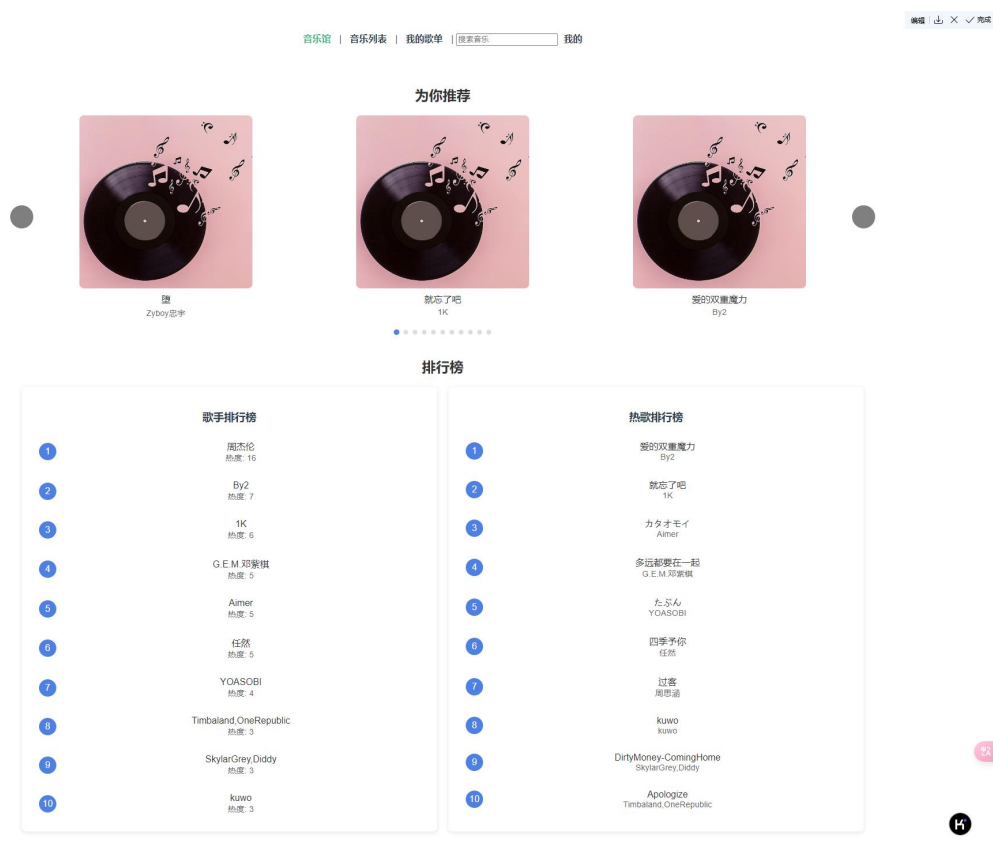


图 42 首页

4. 歌曲详情，包括了播放、加入播放队列、收藏、评论、打分功能，同时还展示了歌曲的详情信息：



图 43 歌曲详情界面

5. 编辑音乐信息：同时，针对管理员，会出现“编辑音乐信息”按钮。

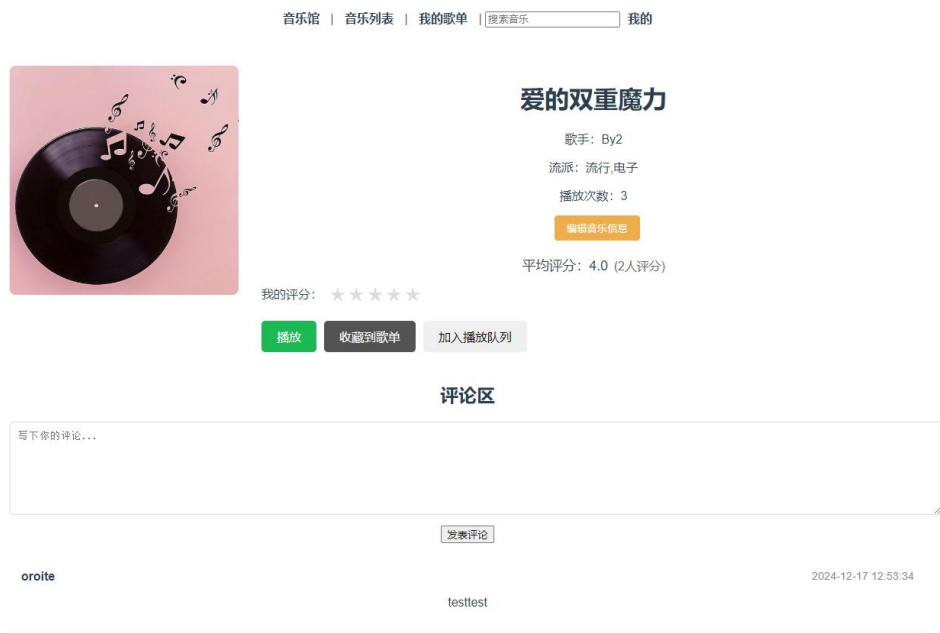


图 44 管理员歌曲详情界面

编辑音乐信息

标题

爱的双重魔力

歌手

By2

流派

流行, 电子

封面图片

选择图片

未选择文件

保存更改

取消

图 45 编辑音乐信息

6. 播放歌曲：

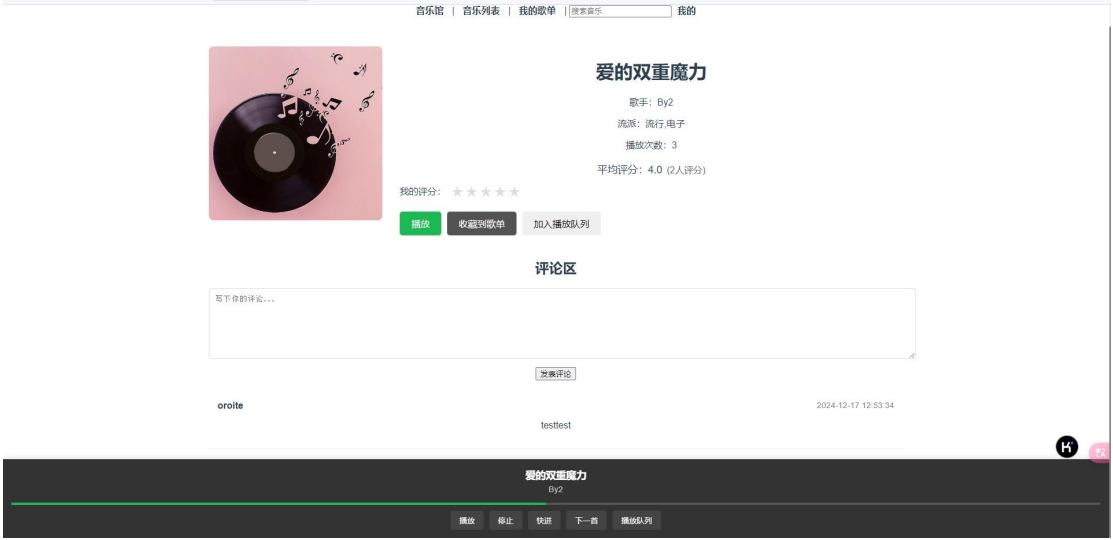


图 46 播放音乐

7. 播放队列，可以选择歌曲播放或者删除，也可以清空播放队列。播放队列存储在 localstage 当中，不会随用户退出而消失：



图 47 播放队列

8. 收藏，可以选择已有歌单和创建新歌单：

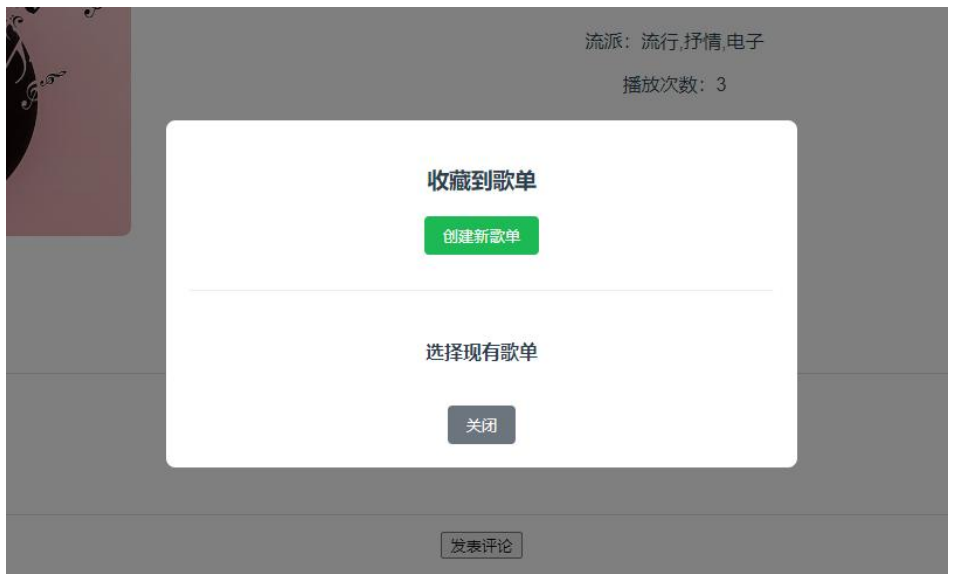


图 48 收藏歌曲



图 49 收藏歌曲时创建歌单



图 50 收藏成功弹窗提示



图 51 创建后收藏界面更新

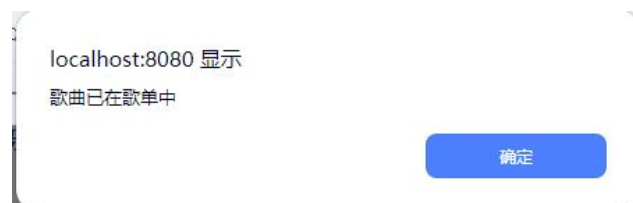


图 52 重复收藏提示

9. 音乐列表，自动从数据库当中获取数据，与用户无关。点击歌曲会自动跳转歌曲详情界面：



图 53 音乐列表

10. 导入音乐信息，针对管理员，会出现“导入音乐”按钮：



图 54 管理员音乐列表出现导入音乐按钮

导入音乐

歌曲名称*

歌手名称*

歌曲风格

封面图片

选择文件

未选择任何文件

音乐文件*

选择文件

未选择任何文件

确认

取消

图 55 导入音乐界面

11. 歌单列表：

音乐馆 | 音乐列表 | 我的歌单 | 搜索音乐

我的

我的歌单

创建歌单

爱的魔力转圈圈

test

创建时间：2024-12-17 21:36:28

删除歌单

图 56 我的歌单列表

localhost:8080 显示

确定要删除这个歌单吗？

确定

取消

图 57 删除歌单

A form titled "创建新歌单" (Create New Playlist). It contains two input fields: "歌单名称" (Playlist Name) and "歌单描述 (可选)" (Playlist Description (Optional)). At the bottom right, there are two buttons: "创建" (Create) in green and "取消" (Cancel) in grey.

图 58 创建歌单

12. 歌单具体信息，可以播放、添加到播放队列、删除：

A card representing a playlist item. At the top, it shows the song title "爱的魔力转圈圈" and the artist "test". Below this, it says "创建时间: 2024-12-17 21:36:28". On the left is a small album cover image. In the center, it displays the song title "多远都要在一起" and the artist "G.E.M.邓紫棋". On the right, there are three buttons: "播放" (Play) in green, "添加到播放队列" (Add to Playlist) in green, and "删除" (Delete) in red.

图 59 歌单歌曲展示

13. “我的”界面，个人具体信息，包括了用户名展示更改、密码修改、音乐喜好设置、个人信息统计：



图 60 个人信息展示界面



图 61 修改用户名界面



图 62 用户名修改提示

图 63 密码修改界面

图 64 音乐喜好设置修改

图 65 偏好修改弹窗提示

14. 管理员个人信息界面，会将普通用户更改成为管理员：

图 66 管理员个人信息

15. 搜索功能，支持模糊查询。输入后回车即可。同样支持点击去往歌曲详情界面：



图 67 搜索结果展示

5.2 系统值得改进（没有实现，需要后面如何规划实现）

1. 音乐推荐算法的优化：

- 当前的协同过滤算法较为简单，可以考虑引入基于内容的推荐
- 增加时间衰减因子，让最近的行为有更高权重
- 引入音乐标签系统，实现更精准的音乐分类和推荐
- 考虑用户的听歌时段偏好进行推荐

2. 用户体验优化：

- 增加音乐播放器的功能，如循环播放、随机播放等
- 添加音乐可视化效果
- 支持歌词显示和同步
- 增加播放列表的拖拽排序功能
- 添加音量调节功能

3. 社交功能的扩展：

- 添加用户关注系统
- 实现歌单分享功能
- 增加评论回复和点赞功能
- 添加用户间的私信功能

4. 安全性增强：

- 实现 JWT token 认证机制

- 增加请求频率限制
 - 加强密码安全性要求
 - 实现敏感操作的二次验证
5. 性能优化：
- 实现数据库查询的缓存机制
 - 优化大量数据的分页加载
 - 优化推荐算法的计算效率
6. 后台管理系统：
- 开发完整的管理员后台界面
 - 增加用户管理功能
 - 添加数据统计和分析功能
 - 实现音乐批量导入功能

这些改进将使系统更加完善和专业，提供更好的用户体验。

5.3 心得体会

在开始，分别了解了 Django 和 Flask 的区别。了解到虽然 Django 的社区十分完善，先行实现了很多功能，但是也是由于其功能众多，不适合初学者进行项目开发，于是选择了相对更为简单的 Flask。同时，通过了解，得到了好用的数据库连接库 SQLAlchemy，并尝试按照推荐的格式完成 `config.py` 和 `models.py` 的实现，最终由于其简单的代码为我的开发节省了很多时间。

同时，前端 Vue 的配置、开发也与 html 开发并不完全相同，也花费了我一定的时间去适应其开发环境。

在开发的过程当中，我获得了宝贵的实践经验。我开始了解到了框架开发与平时开发的一个大不相同，也了解到了微服务运行在多个端口上究竟是什么概念，为什么微服务可以提高服务器的运行效率。项目从概念到实现的每一步都要求我将所学的理论知识与实际应用紧密结合。

在系统设计阶段，我采用了模块化的设计方法，将系统分为用户认证、音乐管理、歌单管理、推荐系统等多个模块。数据库设计上，创建了用户表、音乐表、歌单表等多个关联表，为系统提供了完善的数据支持。同时最终确认使用 Python+Flask+Vue 的技术栈构建了前后端分离的系统，通过 RESTful API 实现数

据交互，这种架构不仅提高了开发效率，也使系统更易于维护和扩展。

在实现的过程中，也遇到了不少的挑战，例如密码加密算法的解决、协同过滤算法的实现、前后端统一（json 格式）的熟悉、音乐播放器实现等。同时，我还需要设置前端界面的不重复出现，需要完成尽可能独立的界面设计，还需要实现 index.js 的路由配置和 App.vue 的开发完成等一系列前端所面临的问题。通过不断的测试和优化，我逐步克服了这些难题，并在此过程中提升了我的编程技能和解决问题的能力。

通过这次项目，我开始深入了解了 Python Web 的开发流程，熟悉了 Flask 和 Vue 的使用开发，开始熟悉协同过滤算法的实现等，有了不小的收获。

当然，虽然我基本实现了此项目，但正如先前提到的“系统值得改进（没有实现，需要后面如何规划实现）”部分所展现的那样，想要真正完成这个系统还需要不小的工作量。

通过这次课程设计，我不仅提升了自己的技术实践能力，也对如何将技术应用于实际问题有了更深的理解。我希望在未来的学习和工作当中，这次项目开发的经验可以很好地帮助我熟悉、掌握新框架的使用，能够让我快速提升自己的技术实力。

参考文献

- [1] 是白小羽呐. 从 0 到 1，Flask 全网最全教学！全文 1w 字，蓝图、会话、日志、部署等使用 Flask 搭建中小型企业级项目[EB/OL]. (2024-09-18)[2024-12-23]. <https://www.cnblogs.com/xiaoyus/p/18418612>.
- [2] 0 世界和平 0. Flask 项目搭建及部署（完整版！全网最全）[EB/OL]. (2019-11-11)[2024-12-23]. <https://blog.csdn.net/asd529735325/article/details/103011940>.
- [3] 逆境清醒. Vue3 安装配置、开发环境搭建(组件安装卸载)（图文详细）[EB/OL]. (2024-12-05)[2024-12-23]. https://blog.csdn.net/weixin_69553582/article/details/129584587.
- [4] Vue.js 中文文档. Vue.js 中文文档[EB/OL]. [2024-12-23]. <https://cn.vuejs.org/guide/introduction>.

[5] Sherry Tian. **【Vue3 项目实战系列一】**—— 从零开始一个 vue3 项目 vue3+ javascript+vite 非常详细 手把手教学[EB/OL]. (2024-12-09)[2024-12-23]. <https://blog.csdn.net/misstianyun/article/details/144178813>.