



SRS Document Heart Wave

波动你的心弦

Wave your heartstrings

Team member :

2051498 储岱泽

2051828 莫益萌

2151298 杨滕超

2151299 苏家铭

2151300 王蔚达

目录

一、简要描述	3
1.1 项目目的	3
1.2 项目范围	3
1.3 术语表	4
1.4 系统设计的改进	4
1.5 实现平台与构架	5
1.6 构架风格	5
1.7 设计模式	7
1.8 关键的设计抉择	9
二、构架完善	10
2.1 平台相关构架	10
2.2 子系统和接口	12
2.3 接口规范（外部）	16
三、设计机制	22
四、用例实现	28
五、产品原型进展	33
5.1 前端原型	33
六、自我反馈	36
七、成员贡献	37

一、简要描述

1.1 项目目的

本系统设计模型文件（SDM 文件）旨在提供详细信息 HW 的设计模型（Wave your heartstrings）。该文件将涵盖体系结构和具有特定接口的相关子系统。不仅如此，文档还给出了系统原型设计的机制和一些实例。

1.2 项目范围

心波 HW 是一个基于 Web 的音乐治愈软件，我们 APP 的目标用户是在日常生活中在意自己心理健康情况的群众，以及音乐、随笔爱好者。我们希望用户可以通过我们的平台更多的关注到自己心理健康以及情绪的变化，通过倾听我们利用情绪识别算法针对情绪推荐的音乐，从而在每一天的生活中保持好心情，提高生活质量与社交质量。用户同样可以通过我们的平台时刻关心身边一些情绪不太好的朋友们，主动给予他们及时的关心，让人间多一份温暖与爱心。

用户可以通过 PC 或移动设备通过浏览器使用我们的网页端程序，或者以手机 App 的形式进行访问。这个平台中的潜在场景包括检索和接收系统推荐的音乐、浏览情绪音乐榜单、发布随笔、建立或加入音乐室等等。用户在使用软件时可以作为访客浏览已设为公开状态的他人的空间，也可以成为该软件社群中的一员发布心情纸条，或是建立音乐室与更多志同道合的朋友一起相互抚慰心灵、交流音乐。



不仅如此，HW 还具有以下特点：

- 运行在网络环境下
- 需要非游客的用户进行实名认证以更精准地建立用户画像
- 拥有集成式数据库
- 具有高迭代度的推荐算法

1.3 术语表

中文名词	英文名词	术语解释
心波	Heart Wave(HW)	软件的名字，意味着“波动你的心弦”。
音乐室	Music Room	一个具有社群功能的群组，用户可以选择创建或加入音乐室并在里面唱歌或交流。
访客	Visitor	对该软件的需求仅仅停留在浏览音乐榜单和收听音乐的一部分用户群体。
用户	User	权限大于访客的群体，可以使用该软件的所有功能。
管理员	Administrator	对各项事务进行管理的角色
群主	Group leader	可在音乐室中管理播放音乐、接收群员等事宜。
随笔	Informal essay	用户可以发布自己的随笔，以便于软件进行情绪识别。
空间	Zones	用户可以在自己的空间发布随笔，其他用户可以在空间看到设置公开的随笔。
打卡	Clock in	用户上线时会进行心情打卡，以便以算法进一步分析用户的心情。

1.4 系统设计的改进

在之前的文档中，我们已经设计了一个 HW 学术支援平台的分析模型。我们在先前的部分里提供了项目的具体用户界面、架构和类图。

在之前的工作基础上，这次我们确定并进一步完善了项目的微服务架构。我们将提供具体的技术架构和逻辑架构模型，并为项目设计具体的接口，详细描述部分第三方 API、子系统接口、分析机制、用例实现和系统原型设计。我们注重项目的具体设计，本次文档说明项目实施所需的主要技术，并确定项目的主要实现方式。我的平台设计将主要有如下的改进：

1. 通过分析平台机制和功能，对逻辑结构等内容进行了重新配置。

2. 除了逻辑架构外，我们进一步增加了技术和物理架构的设计。提供了每一层所需的技术栈和在物理层面上的部署方式。

3. 增加了具体的接口设计。我们根据我们的新子系统提供了具体的接口。其中包含了更多的细节，如参数类型和返回值类型等，考虑了更细节的部分。我们还为第三方和一些子系统的接口提供了详细的说明。

4. 我们使用了 API 网关、数据持久化和一系列的设计机制来提高项目的安全性和可靠性、高容错性和灵活性。

5. 我们对模型中的机制进行分析，并选择合适的设计机制。

6. 在系统的分析设计过程中应用了设计模式。

7. 我们亲自实践了部分项目内容的建设，进行了一部分系统的开发工作，并提出了原型设计。

8. 将用户用例与平台的系统结构结合起来以分析系统的实现过程和方式。

1.5 实现平台与构架

在实际开发中，我们将使用微服务架构。在微服务架构中，系统中的每个微服务都可以独立部署，而且每个微服务都是松散耦合的。每个微服务只专注于一项任务，并对此进行最好的实现。这样可以做到低耦合，更灵活，有针对性地解决问题，更容易独立开发，以及能够带来高可用性和稳定性。为此，我们需要一系列成熟的技术和框架来构建系统。对于整个系统中的每一个子系统和组件，实际

实现的概况大致如下：

- **web 应用：**通过 Vue3、Material-UI、Panache、Supernova、Sylph、Codemagic 以及一些其他工具构建前端框架。使用 AJAX 来更新页面上的信息而不刷新页面。
- **API 网关：**通过应用 Spring Cloud Alibaba，将 API 网关作为一个统一的外部接口来提供微服务。Spring Cloud Alibaba 包含了开发分布式应用微服务的必要组件，这样我们就可以通过 Spring Cloud 编程模型轻松地使用这些组件来开发分布式应用服务。
- **安全：**通过 Sa-Token 进行授权认证，并基于这个轻量级的授权认证框架进行安全测试，以此满足 HW 学术支持平台系统的开发要求。
- **数据存储：**数据存储方面我们将使用关系型数据库 MySQL，文档型数据库 MongoDB，对象存储 MinIO，使用 Redis 来满足高并发情况下数据的安全性和一致性。服务端框架：Spring boot 作为一个强大的网络开发框架，具有高扩展性和出色的性能，我们将使用 Java 语言进行后端开发，并应用 Spring boot 作为后端框架。

1.6 构架风格

系统整体采用微服务架构风格，实现了设计的去中心化，提高了系统的容错性、可扩展性和可维护性。微服务架构的理念不是开发一个巨大的单体应用，而是将应用分解成小型的、相互联系的微服务。一个微服务完成一个特定的功能，这个系统将整个系统划分为五个广泛的微服务子系统部分。

由于 HW 提供的服务是相对完整和独立的，可以打包成不同的微服务集群用于其他系统，因此本系统可以设计成微服务架构系统，具有高内聚、低耦合、不同服务独立部署的特点，同时微服务开发具有弹性，能快速迭代，支持快速更新，可以满足当前系统的需求。

在微服务架构中单个应用的设计过程中，我们的分析将在下面展示。

Garlan 和 Shaw 将软件架构风格分为五类，数据流风格、调用/返回风格、独立组件风格、虚拟机风格和仓库风格。其中：

- 数据流风格包括：批处理序列、管道/过滤器；
- 调用/返回风格包括：主程序/子程序、面向对象风格、层次结构；
- 独立构件风格包括：进程通讯、事件系统；
- 虚拟机风格有：解释器、基于规则的系统；
- 仓库风格有：数据库系统、超文本系统、黑板系统。

我们的系统主要采用调用/返回的方式来设计整体系统。在整体架构方面，基于数据抽象和面向对象的架构，层次结构方面，由小到大进行设计。

1. 数据抽象和面向对象架构

数据抽象和面向对象架构风格的组成部分是对象，对象是抽象数据类型的实例。在抽象数据类型中，数据的表示和相应的操作被封装起来。对象的行为体现在它的接受和请求动作中。连接器是对象之间互动的方式。对象通过函数和程序的调用进行交互。对象是被封装的，一个对象的变化不会影响其他对象。对象有状态和操作，也负责维护状态。这种结构风格包括封装、交互、多态性、集成和重用等特征。为了实现低耦合、高度可重用和可维护的系统，易于升级和保护内部完整性，HW 平台使用 IOC（反转控制）来构建容器，以实现对象之间的解耦和 AOP(Tangent Oriented Programming)来提取被封装为公共服务的公共行为，以减少系统的冗余和模块间的耦合，并提高系统的可操作性和可维护性。

2. 层次结构

分层系统被组织成一个分不同层级的结构。组件在一些层中实现了一个虚拟机。连接器是由决定各层如何互动的协议来定义的。拓扑约束包括对相邻层之间互动的约束。每一层都为上一层提供服务，当使用下一层的服务时，你只能看到与之相邻的一层。大问题被分解成许多渐进的小问题，这些问题被一步步解决，隐藏了很多复杂性。当一个层被修改时，最多影响两个层，通常只有上层能受到影响。上层必须知道下层的身份，不能调整层与层之间的顺序。在本系统以前的设计中，我们采用了如图所示的层次结构。前端部分对应传统三层架构的展示层，而后端架构则是基于微服务的架构。每个微服务都拥有自己的架构，各自通过内网进行通信。其中间引入了中间派发和负载均衡，在底层进行子系统的划分，通过业务逻辑注册控制进行解耦。在本文档涉及的内容中，我们基本保持了这一架构设计思想，并进行了一定的细节完善和改进。

1.7 设计模式

我们平台的开发遵循以下设计模式：

单例模式

单例模式确保一个类只有一个实例，并提供一个全局访问点。

在我们的音乐推荐系统中，我们会使用单例模式来管理全局的配置信息，如数据库连接配置，这样我们可以在整个应用中共享同一份配置信息。此外，音乐播放器也是一个单例，因为在整个应用中我们只需要一个音乐播放器实例。

在我们的音乐推荐系统中，单例模式被应用在登录控制器上，以确保在整个系统中只有一个登录控制器实例。这样可以避免因创建多个登录控制器实例而导致的资源浪费和潜在的错误。以下是单例模式在登录控制器中的应用：

```
public class LoginManager {
    private static LoginManager instance = null;

    private LoginManager() {
        // 私有化构造方法，防止外部创建新的实例
    }

    public static synchronized LoginManager getInstance() {
        if (instance == null) {
            synchronized (LoginManager.class) {
                if (instance == null) {
                    instance = new LoginManager();
                }
            }
        }
        return instance;
    }
}
```

在这个例子中，`LoginManager` 是我们的登录控制器。我们在 `LoginManager` 类中定义了一个静态成员 `instance`，这个成员是 `LoginManager` 类型的，它用来保存 `LoginManager` 的唯一实例。我们将 `LoginManager` 的构造方法私有化，这样外部就不能直接创建 `LoginManager` 的新实例。然后我们提供了一个公共的静态方法 `getInstance`，这个方法用来获取 `LoginManager` 的唯一实例。在 `getInstance` 方法中，我们首先检查 `instance` 是否为 `null`，如果是，那么我们就创建一个新的 `LoginManager` 实例并赋值给 `instance`；如果不是，那么我们就直接返回 `instance`。注意，我们在创建 `LoginManager` 实例的过程中使用了 `synchronized` 关键字，这是为了防止在多线程环境下创建多个 `LoginManager` 实例。我们首先在方法级别上使用了 `synchronized`，这样在同一时间只能有一个线程进入 `getInstance` 方法。然后我们在创建 `LoginManager` 实例的代码块上再次使用了 `synchronized`，这样在同一时间只能有一个线程执行这个代码块。这种双

重检查锁定的方式可以确保在多线程环境下 `LoginManager` 的唯一性。通过这种方式，我们可以确保 `LoginManager.getInstance()` 总是返回全局唯一的登录控制器实例。

职责链模式

职责链模式是一种行为型设计模式，它创建了一个对象链。这个链上的每个对象都有机会处理请求，直到某个对象处理它为止。这种模式在处理流程中非常有用，因为它可以使多个处理逻辑不同的对象能够处理请求，并将这些对象连成一条链，沿着这条链式结构传递请求，进行请求的处理。

在我们的音乐推荐系统中，我们可以使用职责链模式来处理各种不同的用户请求。例如，我们可能有一个请求处理链来处理用户的情绪分析记录请求、个性化情绪音乐推荐请求、音乐室社交系统的请求等。每个处理器在链中都有机会处理请求，如果它不能处理，就将请求传递给链中的下一个处理器。这样，我们可以将处理逻辑分散到各个处理器中，使得系统更加灵活和可扩展。

此外，职责链模式也可以帮助我们避免请求的发送者与处理者之间的耦合。在我们的系统中，请求的发送者不需要知道请求是由哪个处理器处理的，它只需要将请求发送到处理链上，然后由处理链上的处理器来决定谁来处理这个请求。这样，我们可以在不改变请求发送者的代码的情况下，动态地改变请求的处理逻辑，增加或删除处理器，这使得我们的系统更加灵活和易于维护。

例如，我们可能有一个处理用户情绪分析记录请求的处理器，这个处理器在链中的位置可能会根据系统的需求和设计而变化。如果我们发现用户情绪分析记录请求的处理逻辑需要改变，我们可以很容易地在处理链中添加一个新的处理器来处理这个请求，而不需要修改请求的发送者的代码。

观察者模式

观察者模式定义了对象之间的一对多依赖关系，当一个对象的状态改变时，所有依赖于它的对象都会得到通知并自动更新。

在我们的系统中，我们会使用观察者模式来实现事件驱动的功能。例如，当用户的情绪得分改变时，我们需要通知其他系统组件进行相应的更新，如个性化情绪音乐推荐系统需要根据用户的新的情绪得分来更新推荐的音乐。

代理模式

代理模式为其他对象提供一个代理或占位符，以控制对这个对象的访问。在我们的系统中，我们可能会使用代理模式来控制对某些资源的访问。例如，我们会使用代理模式来实现网络请求的缓存，以提高系统的性能。我们可以创建一个音乐请求的代理类，这个类在接收到音乐请求时，首先检查缓存中是否有请求的结果，如果有，就直接返回缓存的结果，如果没有，就执行网络请求，然后将结果存入缓存。

策略模式

策略模式定义了一系列的算法，并将每一个算法封装起来，使它们可以互相替换。策略模式让算法的变化独立于使用算法的客户。

在我们的系统中，我们可能会使用策略模式来实现不同的业务逻辑。例如，我们可能会有不同的策略来处理不同类型的用户请求。每个策略都封装了一种处理请求的算法，我们可以在运行时根据请求的类型来选择使用哪种策略。例如，我们可能有一个策略用于处理用户的情绪分析记录，另一个策略用于处理个性化情绪音乐推荐，另一个策略用于处理音乐室社交系统的请求，等等。这样，我们可以将不同的处理逻辑分散到各个策略中，使得系统更加灵活和可扩展。

解释器模式

解释器模式为给定的语言定义了一个表示，并定义了一个解释器，这个解释器使用该表示来解释语言中的句子。

在我们的音乐推荐系统中，解释器模式可能会被用于处理用户的搜索请求或者其他复杂的用户输入。例如，用户可能会输入一些复杂的查询，如“我想听一些快乐的音乐”或“我需要一些帮助我放松的音乐”。这些查询可能包含了一些复杂的语义，我们需要一个解释器来解析这些查询，然后将它们转换成系统可以理解的形式。

具体来说，我们可以定义一个 `MusicQuery` 接口，这个接口定义了解析音乐查询的方法。然后我们可以创建一个 `MusicQueryInterpreter` 类，这个类实现了 `MusicQuery` 接口，它可以解析用户的音乐查询，并将它们转换成系统可以理解的形式。当用户输入一个音乐查询时，我们的系统就可以创建一个 `MusicQueryInterpreter` 实例，然后使用这个实例来解析用户的查询。这样，我们就可以根据用户的查询来推荐适合的音乐，而不需要修改系统的内部表示。

这就是解释器模式在我们的音乐推荐系统中的一个应用实例。通过使用解释器模式，我们可以将复杂的查询解析逻辑抽象化，使得系统更加灵活和可扩展。同时，解释器模式也使得我们的代码更加清晰和易于理解，提高了代码的可读性和可维护性。

1.8 关键的设计抉择

- 使用 `Vue3` 和 `MaterialUI` 构建我们的 UI: `Vue3` 作为新一代的跨平台前端框架，支持以原生的方式在各平台上进行渲染；这样在提高了前端开发的效率的同时，也使得用户具有更加流畅的体验。`MaterialUI` 的引入使得前端设计更加规范合理。
- 使用微服务架构：微服务架构具有较大的弹性，可以方便地与敏捷开发过程相结合，在重构和持续集成时具有传统架构不可比拟的优势。此外，微服务架构易于部署和扩展的特性使得我们在运维的过程中可以利用云平台有效控制成本。
- 尊重用户隐私：我们在构建系统的各个部分时都将保护用户隐私作为首要任

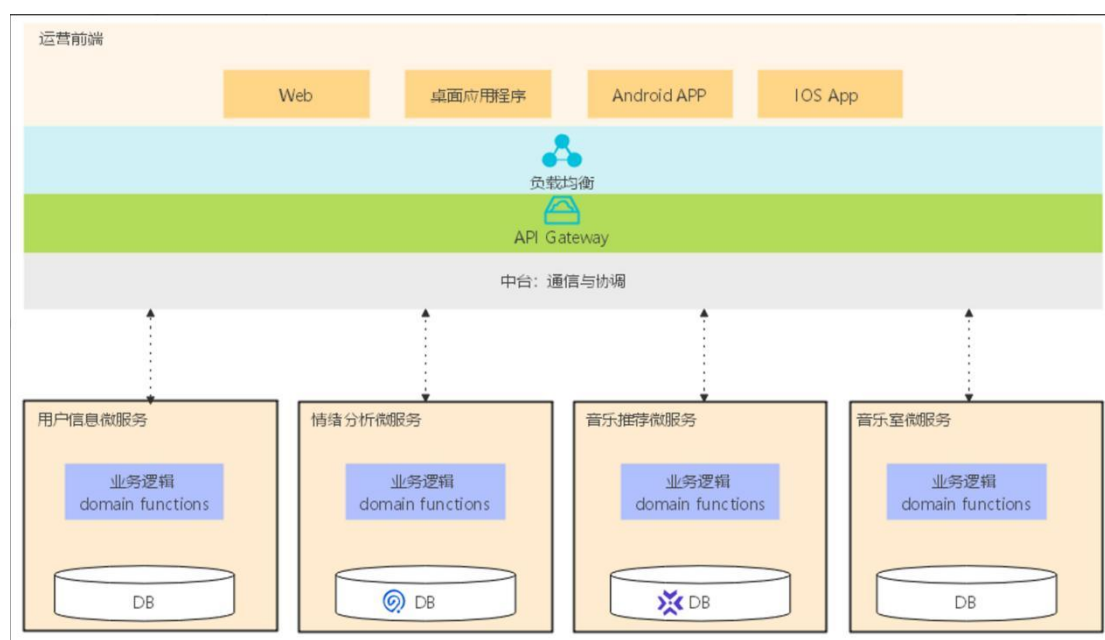
务，除了基于 sa-token 的权限控制外，我们也考虑使用数据库层面的加密和磁盘级别的硬件加密，以充分保护用户数据。此外，我们在进行数据采集和分析的过程中会使用已经脱敏后的数据，以保障用户的隐私安全。

- 以 OOP 风格为基础，兼用其它领域特定语言：我们的项目的大部分逻辑是使用 Spring 技术栈开发的，但在一些专用的服务上（例如数据库，数据分析等），受益于微服务框架，我们也采用对应领域的特定技术栈，从而提高开发效率。

二、构架完善

2.1 平台相关构架

我们原先的系统架构图如下：



经过对 HW Music 进行系统性的分析，以及为了确保服务的粒度适中而不冗杂，我们将系统的微服务按照功能进行进一步划分：

- 用户管理微服务
- 音乐管理微服务
- 推荐系统微服务
- 情绪分析微服务
- 社交互动微服务
- 随笔管理微服务

对于改进的微服务架构，我们通过对其相关技术进行分析，对系统使用的技术栈进行选择细化，得到了以下的技术架构图：

HeartWave Music 平台相关架构图



我们的整体架构依然采用微服务架构，每个微服务均有独立的数据库和各自的业务逻辑，可以分别进行独立的开发。

我们的表现层根据所属平台的不同，分成了三个板块进行开发。首先，在 Web 开发当中，我们采用了 Vue3，这是一个高效、灵活的前端库，能够创建可复用的 UI 组件，有利于代码的维护和更新。配合 Redux 作为状态管理器，使得应用状态可预测且易于调试。Material-UI 作为 UI 库，提供了一套丰富的 UI 组件和主题，便于快速开发出美观且符合 Material Design 规范的应用。Webpack 则负责模块打包，实现代码分割、懒加载等功能，有助于提高应用性能。这套技术栈应用于 Web 平台。

对于 IOS/Android 平台，我们采用 React Native 框架进行跨平台应用的开发，一套代码就能实现 iOS 和 Android 的应用，大大提高了开发效率，React Native Elements 作为 UI 库，提供了一整套组件，方便移动端的开发。同时，也使用了 Redux 进行状态管理。

在 Windows/MacOS 平台，我们采用了 Electron 进行桌面应用的开发，

Electron 允许使用纯 JavaScript、HTML 和 CSS 等 Web 技术创建跨平台的桌面应用，大大降低了开发复杂度。Photon 作为 UI 库，专为 Electron 设计，用于创建美观的桌面应用。Jest 作为测试框架，保障代码质量和稳定性。

前端在经过 API Gateway 后，进入到后端的微服务系统。API Gateway 作为系统的入口，可以处理路由、负载均衡、权限验证等问题，保障系统的安全和稳定。

后端使用了 Spring Cloud, Spring AOP, Spring Boot 和 RabbitMQ 等技术栈来开发微服务。Spring Boot 使得 Spring 应用变得更加简单，Spring Cloud 提供了一整套微服务解决方案，Spring AOP 负责系统的日志、事务、权限等横切面，RabbitMQ 作为消息队列，解耦微服务之间的通信。

用户信息微服务：我们使用了 sa-token 和 OAuth-2.0 作为用户验证和权限控制的方式。sa-token 是一种轻量级的权限验证框架，适合国内的开发习惯，同时具有良好的扩展性，为我们提供了灵活的权限控制机制。OAuth 2.0 是一个行业标准的授权框架，可以让第三方应用在用户同意的前提下获得访问的权限，这也增加了我们应用的合作和分享能力。

音乐管理微服务：我们使用了 MinIO 和 Elasticsearch。MinIO 是一个开源的对象存储服务，它允许你将非结构化数据（比如音乐文件）存储为可访问的对象，这对于音乐文件的管理非常有用。Elasticsearch 作为一个高度可扩展的开源全文搜索和分析引擎，使我们能够以近实时的速度进行复杂的搜索操作，例如搜索歌曲名称、歌手、专辑等信息。

推荐系统微服务：我们使用了 Apache Spark 和 Redis。Apache Spark 是一个用于处理大数据的快速、通用、可扩展的大数据处理平台，它提供了包括 SQL 查询、流处理、机器学习等在内的一系列工具，适用于推荐系统的实现。Redis 是一个开源的内存中数据结构存储系统，它能提供极快的读写速度，可以用来存储用户的行为数据，提供给推荐系统进行实时推荐。

情绪分析微服务：我们采用 Tensorflow, Librosa, ONNX 进行情绪分析。Tensorflow 是一个强大的开源库，用于数据流程编程，特别是训练和推理深度学习模型。Librosa 是一个用于音频和音乐分析的 Python 库，非常适合音乐和语音的情绪分析。ONNX 则是一个开放的模型格式，可以使得 AI 模型在不同的平台之间更好地移植和部署。

社交互动微服务：我们使用了 Websocket 和 Redis。Websocket 提供了一种在单个 TCP 连接上进行全双工通信的协议，使得服务器可以实时地推送信息给客户端，非常适合于社交互动的场景。同样，这里也使用了 Redis 来存储和查询用户的社交数据，如好友列表、聊天记录等。

随笔管理微服务：我们同样采用了 Elasticsearch，通过它的全文搜索和分析能力，可以方便地管理和检索用户的随笔。

在数据库层面，所有的微服务都采用了 JDBC, Hibernate, JPA 进行处理。JDBC 是 Java 数据库连接的标准 API，Hibernate 是一个全功能的 JPA 实现，它们一起为我们提供了一套强大的 ORM 解决方案，大大简化了数据库操作，使得我们可以更专注于业务逻辑的实现。

2.2 子系统和接口

根据领域驱动设计的理念，为了更好地设计相应的领域服务，我们将系统涉

及的业务边界按照一定的逻辑进行划分,以更好地组织和管理系统的功能和业务流程。以下是根据我们对系统的理解,划分的几个领域:

- 用户管理领域:** 负责处理用户的注册、登录、权限管理等与用户相关的操作。
- 音乐管理领域:** 涵盖音乐资源的上传、管理、搜索和推荐等功能,以及与音乐相关的标签、分类和专辑管理等操作。
- 推荐系统领域:** 基于用户的情绪和音乐特征,进行个性化的音乐推荐,提供优质的音乐体验。
- 情绪分析领域:** 负责对用户的随笔进行文本情绪分析,从中提取情绪信息,并与音乐进行情绪匹配。
- 社交互动领域:** 包括音乐室聊天、唱歌、一起听歌等社交功能,用户可以与其他用户进行互动和分享音乐。
- 随笔管理领域:** 提供随笔记录、查看好友随笔、评论等功能,为用户提供了表达自己想法的平台。

根据领域设计和为了确保服务的粒度适中而不冗杂,我们将系统的微服务按照功能进行进一步划分:

- 用户管理微服务
- 音乐管理微服务
- 推荐系统微服务
- 情绪分析微服务
- 社交互动微服务
- 随笔管理微服务

用户服务系统

序号	REST API	接口解释
1	POST /api/user/	该接口用于实现用户的注册功能,可以将创建新用户的信息。
2	GET /api/user/?userid=id&userPassword=password/	该接口用来实现用户的登录系统的功能。
3	PUT /api/user/?userid=id/	用户可以通过这个接口向后端发起修改个人信息的请求。
4	GET /api/userinfo/	管理员可以通过该接口获取所有用户的信息,通过该接口返回所有用户和其信息的列表
5	DELETE /api/user/?userid=id/	用户通过该接口发出注销用户账号的请求,返回是否注销成功的信息。
6	POST /api/user/mood/?userid=id/	用户通过该接口上传自己的心情打卡情绪状况,并返回上传是否成功。

音乐管理服务系统

序号	REST API	接口解释
1	GET /api/music/?userid=id/	系统通过该接口获取该用户收藏的音乐和歌单。
2	POST /api/music/	管理员可以通过该接口上传最新的音乐
3	DELETE /api/music/?musicid=id/	管理员可以通过该接口删除一些被要求下架、没有版权或者内容不宜的音乐。
4	PUT /api/music/?musicid=id/	管理员可以通过该接口更新一些音乐的信息。
5	PUT /api/music/songList/?songListID=id	用户通过该接口修改更新自己已有歌单中的音乐，并返回修改信息。

音乐推荐服务系统

序号	REST API	接口解释
1	GET /api/recommendations/?userid=id	该接口用来返回推荐给该用户的音乐列表，这些音乐会出现在用户的音乐推荐界面上。

情绪分析服务系统

序号	REST API	接口解释
1	POST /api/emotion-analysis/	该接口将用户的日记内容上传进行文本情感分析，同时上传用户每日心情打卡的信息，将两者分析出的心情值进行加权计算，然后返回情绪分析结果。
2	GET /api/emotion-analysis/history	用户通过该接口获取过去情绪分析的记录，并且形成心情周报。

社交互动服务系统

序号	REST API	接口解释
1	GET /api/friends/?userid=id	用户通过该接口获得自己的好友列表。
2	GET/api/musicRoom/?userid=id	用户通过该接口获取自己加入的音乐室列表。
3	POST /api/friends/chat/send/?friendid=id	用户通过该接口向好友发送消息，进行聊天，返回发送成功或者失败的信息。
4	POST /api/musicRoom/chat/send/?roomid=id	用户通过该接口向指定音乐室发送消息，进行聊天，返回发送成功或者失败的信息。
5	POST /api/friends/add/?friendid=id	用户通过该接口添加好友，返回发送成功或者失败的信息。
6	POST /api/musicRoom/add/?roomid=id	用户通过该接口申请加入音乐室，返回发送成功或者失败的信息。
7	POST /api/music-room/sing/?roomid=id&?userid=id&songid=id	用户通过该接口申请在指定音乐室内唱歌。
8	POST /api/music-room/listen/	用户通过该接口请求在音乐室中“一起听歌”
9	POST /api/music-room/	用户通过该接口创建新的音乐室。
10	POST /api/music/comment/?musicid=id/	用户可以通过该接口对音乐进行评论。返回是否操作成功。
11	GET /api/comments/?musicid=id/	用户可以通过该接口获取该音乐的评论列表，返回是否操作成功。
12	POST /api/comments/?musicid=id/like	用户通过该接口对音乐进行点赞并收藏。返回是否操作成功。
13	DELETE /api/comments/?musicid=id/like	用户通过该接口取消对音乐进行的点赞与收藏。返回是否操作成功。

随笔管理服务系统

序号	REST API	接口解释
1	POST /api/diary	用户通过该接口上传日记。返回是否操作成功。
2	GET /api/diary/?diaryid=id	用户通过该接口获取指定日记的内容。返回是否操作成功。

3	PUT /api/diary/?diaryid=id	用户通过该接口更新指定日记的内容,返回修改信息。
4	DELETE /api/diary/?diaryid=id	用户通过该接口删除指定的日记,返回是否删除成功。

2.3 接口规范（外部）

在为了提高开发效率和降低成本的考虑下,我们选择了一系列第三方 API 来构建我们的服务。在选择外部 API 时,我们主要考虑以下几个方面:

- 1. 综合成本:** 我们确保使用第三方 API 的综合成本不会高于自行开发相同功能的成本。
- 2. 服务的可靠性与稳定性:** 我们选择的第三方 API 必须具备稳定的接口功能和可用性,确保服务的稳定运行。
- 2. 回退计划:** 我们制定了相应的回退计划,以应对第三方 API 临时不可用的情况,确保系统的可靠性和可用性。
- 3. 引入竞品:** 针对具有相似功能的第三方 API,我们引入了至少两个供应商的 API,以增加系统的可靠性和容错能力。

基于上述考虑,我们在以下领域引入了第三方 API,并对其进行了适当的封装以方便复用:

1、音乐数据 API: 我们使用了音乐数据提供商的 API,以获取丰富的音乐数据和相关信息,用于音乐管理微服务和推荐系统微服务:

- **网易云音乐 API:** 网易云音乐作为一家知名的音乐平台,拥有大量的正版音乐资源,可以通过 API 获取丰富多样的音乐曲目、专辑和艺术家信息。
- **Spotify API:** 提供音乐流媒体服务和音乐相关信息的 API,可以获取音乐曲目、专辑、艺术家信息等。
- **Last.fm API:** 提供音乐推荐和音乐元数据的 API,可以获取音乐推荐、相似艺术家、歌曲标签等信息。

2、自然语言处理 API: 为了进行情绪分析和文本处理,我们采用了自然语言处理 API,用于情绪分析微服务中的文本情绪分析功能:

- **Google Cloud Natural Language API:** 提供自然语言处理功能的 API,可以进行文本情绪分析、实体识别、关键词提取等。
- **Microsoft Azure Text Analytics API:** 提供文本分析功能的 API,可以进行情感分析、语言检测、实体链接等。

3、社交互动 API: 我们集成了社交互动平台的 API,以实现好友聊天、音乐室聊天、加入音乐室、添加好友等功能,用于社交互动微服务。

- **Facebook Messenger API:** 提供与 Facebook Messenger 集成的 API,可以实现好友聊天、发送消息、创建群组等功能。
- **WebSocket API:** 用于实现实时聊天功能,可以建立双向通信通道,支持音乐室聊天和音乐室一起听歌等互动功能。

4、文件存储和分享 API: 为了支持随笔管理微服务中的随笔记录和分享功能,我们使用了文件存储和分享 API 用于存储和分享用户的随笔内容。

- **Amazon S3 API:** 用于文件存储的云服务 API，可以存储和管理用户随笔的文本和图片等文件。

- **Dropbox API:** 提供文件存储和分享功能的 API，可以实现用户之间的随笔分享和协作。

5、**二次验证 API:** 短信接口，Google 2FA 等。

通过引入这些第三方 API 并进行适当的包装，我们能够快速集成现有的功能和服务，提高开发效率，同时确保系统的可靠性和稳定性。

下面以音乐数据 API 为例，详细介绍其包装方式：

请求样例：

```
{
  "user":{
    "userID":"12535662232",
    "isVIP":true
  },
  "musicName":"Nocturne No. 12 in G, Op. 37, No. 2",
  "musician":"Chopin",
  "musicMood":"Happy"
}
```

返回样例：

```
{
  "NetEase_Cloud_Music_API_response":{
    "code": 200,
    "message": "success",
    "data": {
      {
        "id": "1356623E2BE694F3F081CA688B71150F",
        "url":"http://m702.music.126.net/20230601192328/8879ad4dd870c00931dad42189f80904/jd-musicrep-ts/ebd7/04e4/31f0/20cfa4d0d7a0687914ca2",
        "size": 1202721,
        "md5": "20cfa4d0d7a0687914ca23d2e61fc159",
        "songName":"Nocturne No. 12 in G, Op. 37, No. 2",
        "artist":"Chopin",
        "comments":[]
      }
    }
  }
}
```

异常样例：

```
{
  "NetEase_Cloud_Music_API_response":{
    "code": 200,
    "subCode": "10003",
    "message": "请求失败",
    "data": {
      "url": null,
      "size": 0,
      "md5": null,
    }
  }
}
```

通过将复杂参数处理留给后端，我们能够提供更简洁、易用的 API 接口给应用程序使用。这种简化的设计有助于提高开发效率，并降低了前端开发人员对复杂参

数的处理负担。同时，我们在 API 设计中注重安全性，采用了授权系统和签名等技术来确保数据的安全传输和访问权限的控制。此外，我们对异常情况的处理也考虑了极简原则，以保护内部系统的安全性和稳定性，同时为应用程序提供简明的错误信息，提高用户体验。

同时，作为网易云音乐的插件，我们也向网易云音乐提供一个入口，用户通过网易云音乐上的这个入口可以进入到我们 HeartWave 插件的界面，在进入我们的插件界面的时候，我们将会对网易云音乐的用户信息进行读取与验证，以登录到我们插件的界面，在用户企图进入我们的插件界面时，接口调用参数如下所示：

字段	必选	类型	说明
type	FALSE	string	返回数据方式（默认以 json 的方式返回）
appkey	TRUE	string	应用 appkey
ts	TRUE	int	当前时间戳（UNIX TIMESTAMP）
sign	TRUE	string	应用校验密钥
callback	FALSE	string	JSON 调用方式时回调函数名称
access_key	FALSE	string	应用在用户申请登录后获取到的 access_key
platform	FALSE	string	客户端平台信息

一些通用的返回值如下表所示：

代码	说明
-1	插件不存在或者已经被封禁
-2	Access key 错误
-3	API 校验密钥错误
-101	账号未登录
-102	账号被封停
-105	验证码错误
-106	账号未激活
-108	应用没有存取相应功能的权限
-400	请求有误
-403	权限不足
-404	文档不存在
-500	服务器内部错误
-503	调用速度过快

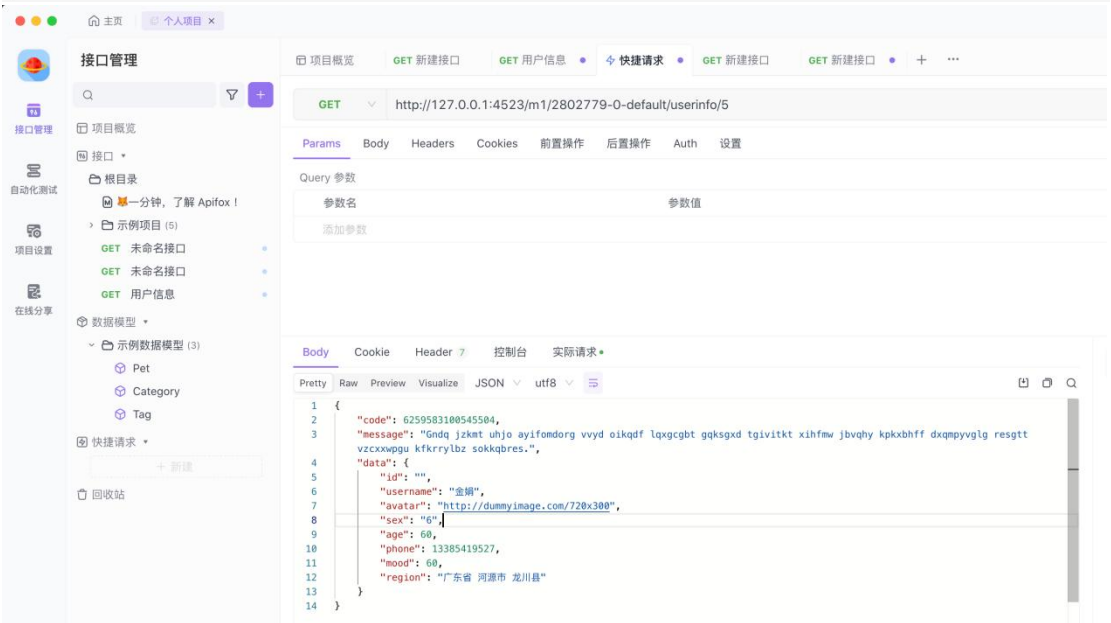
2.4.接口样例

接下来，我们以获取具体用户信息、获取推荐歌曲列表、上传情绪分析数据、获取用户情绪数据四个接口为例，展示我们具体的接口设计与实现。

获取具体用户信息

该请求为 GET 请求，REST API 为/api/user/?userid=id，参数为管理员希望获取到的用户的 id，返回的 JSON 如下所示：

```
{
  "code": 6259583100545504,
  "message": "Gndq jzkmt uhjo ayifomdorg vvyd oikqdf lqxcggbt gqksxd tgivitkt xihfmw jbvqhy kpkxbhff dxqmpyvlg resgtt vzcxxwpgu kfkrrylbz",
  "data": {
    "id": "",
    "username": "金娟",
    "avatar": "http://dummyimage.com/720x300",
    "sex": "female",
    "age": 60,
    "phone": 13385419527,
    "mood": 60,
    "region": "广东省 河源市 龙川县"
  }
}
```



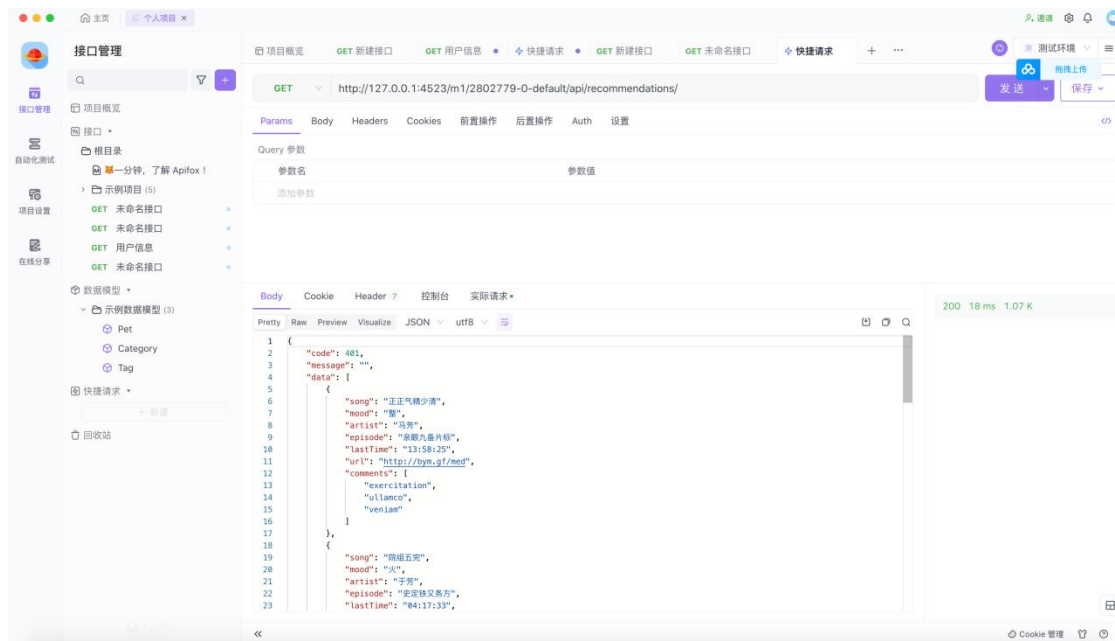
The screenshot shows the Apifox interface. On the left, there's a sidebar with '接口管理' (Interface Management) and a list of interfaces. The main area shows a GET request to 'http://127.0.0.1:4523/m1/2802779-0-default/userinfo/5'. The response body is a JSON object with the following structure:

```
{
  "code": 6259583100545504,
  "message": "Gndq jzkmt uhjo ayifomdorg vvyd oikqdf lqxcggbt gqksxd tgivitkt xihfmw jbvqhy kpkxbhff dxqmpyvlg resgtt vzcxxwpgu kfkrrylbz sokkqbres.",
  "data": {
    "id": "",
    "username": "金娟",
    "avatar": "http://dummyimage.com/720x300",
    "sex": "female",
    "age": 60,
    "phone": 13385419527,
    "mood": 60,
    "region": "广东省 河源市 龙川县"
  }
}
```

获取推荐歌曲列表

该请求为 GET 请求，REST API 为/api/recommendations/?userid=id，参数为请求返回推荐歌曲列表的用户的 id，返回的 JSON 如下所示：

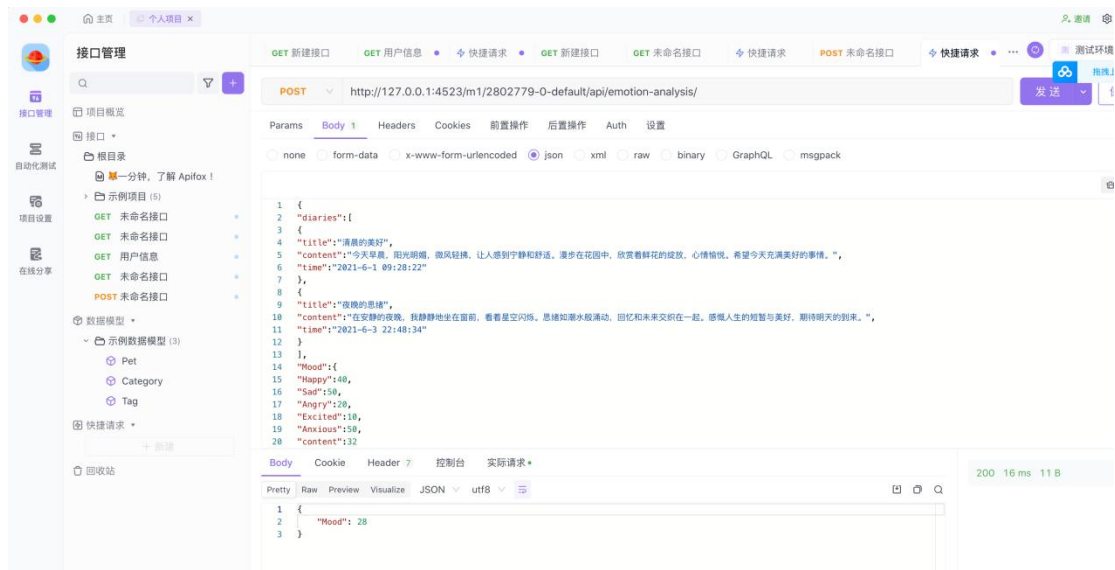
```
{
  "code": 200,
  "message": "success",
  "data": [
    {
      "song": "安九",
      "mood": "Happy",
      "artist": "老王乐队",
      "episode": "吾日三省吾身",
      "lastTime": "5:20",
      "url": "http://m702.music.126.net/20230601192328/8879ad4dd870c00931dad42189f80904/jd-musicrep-ts/ebd7/04e4/31f0/20cfa4d0d7a0687914ca23d2e61fc159.i",
      "comments": []
    },
    {
      "song": "Levitating",
      "mood": "Happy",
      "artist": "Dua Lipa",
      "episode": "Future",
      "lastTime": "3:23",
      "url": "http://m702.music.126.net/20230601192328/8879ad4dd870c00931dad42189f80904/jd-musicrep-ts/ebd7/04e4/31f0/20cfa4d0d7a0687914ca23d2e61fc159.i",
      "comments": []
    }
  ]
}
```



上传情绪分析数据

该请求为 post 请求，REST API 为/api/emotion-analysis/，该接口将用户的日记内容上传进行文本情感分析，同时上传用户每日心情打卡的信息，将两者分析出的心情值进行加权计算，然后返回情绪分析结果，需要以 JSON 格式传递请求 body，格式如下：

```
{
  "diaries":[
  {
    "title":"清晨的美好",
    "content":"今天早晨，阳光明媚，微风轻拂，让人感到宁静和舒适。漫步在花园中，欣赏着鲜花的绽放，心情愉悦。希望今天充满美好的事情。",
    "time":"2021-6-1 09:28:22"
  },
  {
    "title":"夜晚的思绪",
    "content":"在安静的夜晚，我静静地坐在窗前，看着星空闪烁。思绪如潮水般涌动，回忆和未来交织在一起。感慨人生的短暂与美好，期待明天的到来。",
    "time":"2021-6-3 22:48:34"
  }
  ],
  "Mood":{
    "Happy":40,
    "Sad":50,
    "Angry":20,
    "Excited":10,
    "Anxious":50,
    "content":32
  }
}
```

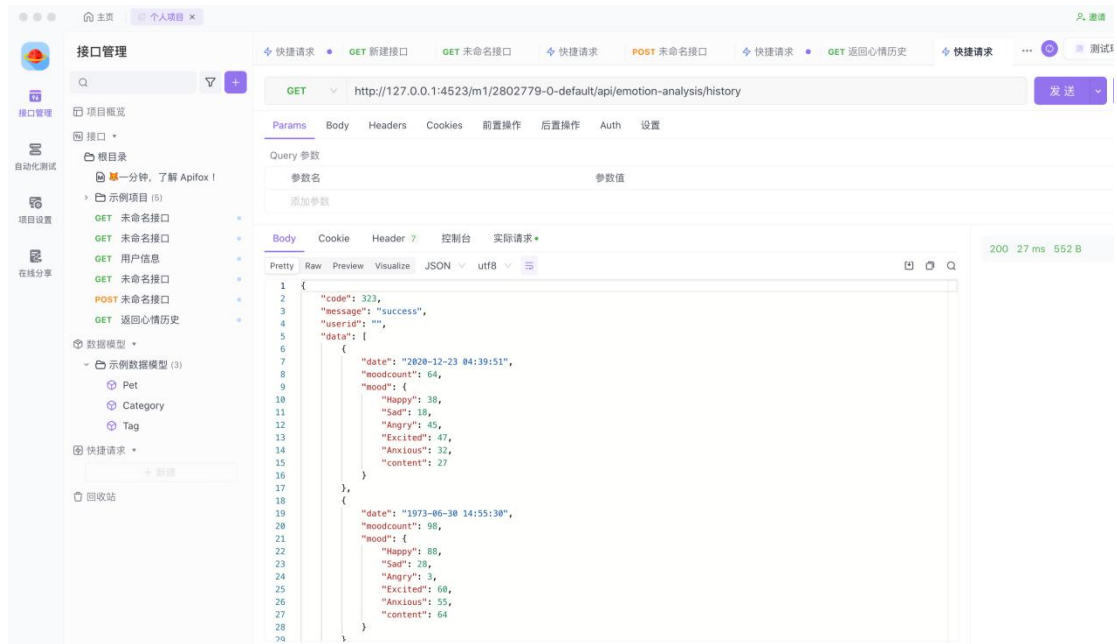


返回心情值为：28。

获取用户情绪数据

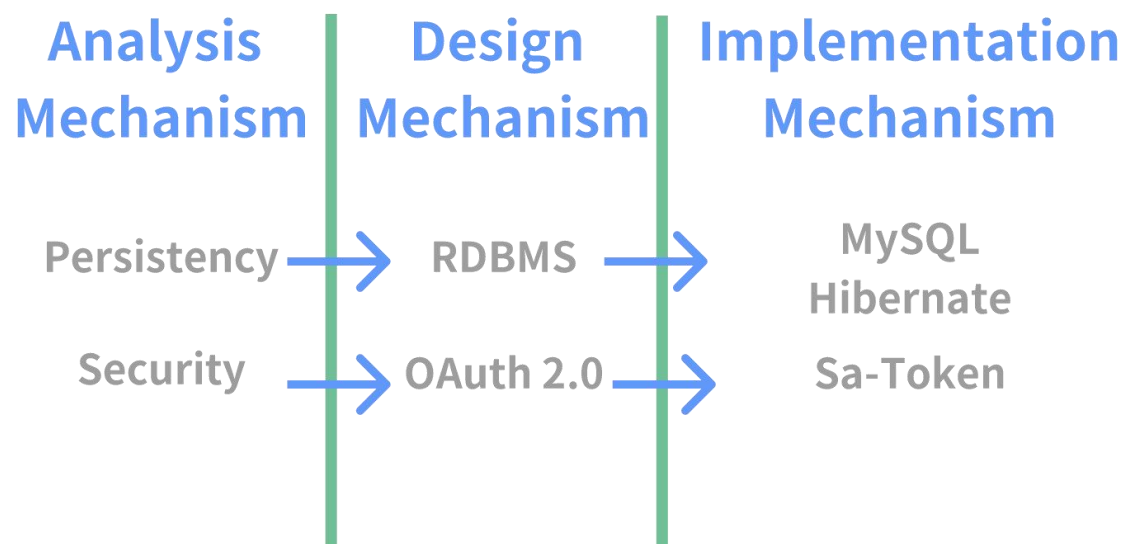
该请求为 GET 请求，REST API 为 `/api/emotion-analysis/history`，参数为请求返回心情历史记录的用户 id，返回的 JSON 如下所示：

```
{
  "code": 200,
  "message": "success",
  "userid": "1234434",
  "data": [
    {
      "date": "2012-12-10",
      "moodcount": 23,
      "mood": {
        "Happy": 40,
        "Sad": 50,
        "Angry": 20,
        "Excited": 10,
        "Anxious": 50,
        "content": 32
      }
    },
    {
      "date": "2014-11-12",
      "moodcount": 67,
      "mood": {
        "Happy": 70,
        "Sad": 21,
        "Angry": 11,
        "Excited": 78,
        "Anxious": 21,
        "content": 66
      }
    }
  ]
}
```



三、设计机制

HW 的架构设计如下图所示：



以下将讨论设计机制中的几个案例：

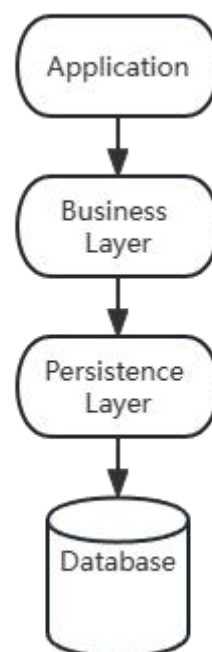
数据持久存储机制

数据持久存储机制是指将数据在计算机系统中长期存储的方法和技术。它的存在使得程序可以在多次运行之间保留数据状态，从而提高了程序的可靠性和效率。其中，数据的存储需要使用内存、硬盘等物理介质。但是，内存中的数据会随着计算机的关闭而消失，而磁盘上的数据可以长期保存。因此，数据持久化机制可以确保数据在程序关闭之后得到长期保留。再者，如果数据只存在于内存中，

那么一旦程序崩溃或电源故障发生，所有未保存的数据都将丢失。通过使用数据持久化机制，可以将数据写入到硬盘上，从而确保即使系统故障或突然关闭时，也可以恢复数据。同时，数据持久化机制还允许多个用户访问相同的数据，这种情况在共享应用程序或数据库时非常常见。如果数据仅保存在一个用户的内存中，则其他用户无法访问该数据。有了数据持久化，多个用户可以同时访问存储在磁盘上的数据。正如其他应用软件一样，我们的 HW 中有许多数据需要进行永久保存，从而实现系统的相关功能。例如保存用户的账户信息、留言板内容、随笔内容甚至更重要的是情绪识别对某一特定用户的识别结果。根据我们的要求，我们需要将系统数据存储入关系型数据库中。

此时，我们需要使用 ORM 将数据库中的关系数据和面向对象编程语言中的对象进行映射。原因在于，在实际编程中，我们将使用面型对象编程语言，从而表达各种实体之间的继承、关联等关系，而不是使用低级别的 SQL 代码手足无措；同时，ORM 提供了一个清晰的抽象层，隐藏了底层的数据库实现细节，使得应用程序更易于维护和升级；再者，ORM 使应用程序更可靠，因为它可以帮助开发人员避免常见的错误，如 SQL 注入攻击等；最后，ORM 框架通常会自动优化生成的 SQL 查询，从而使应用程序更快速响应，并减少数据库访问次数。

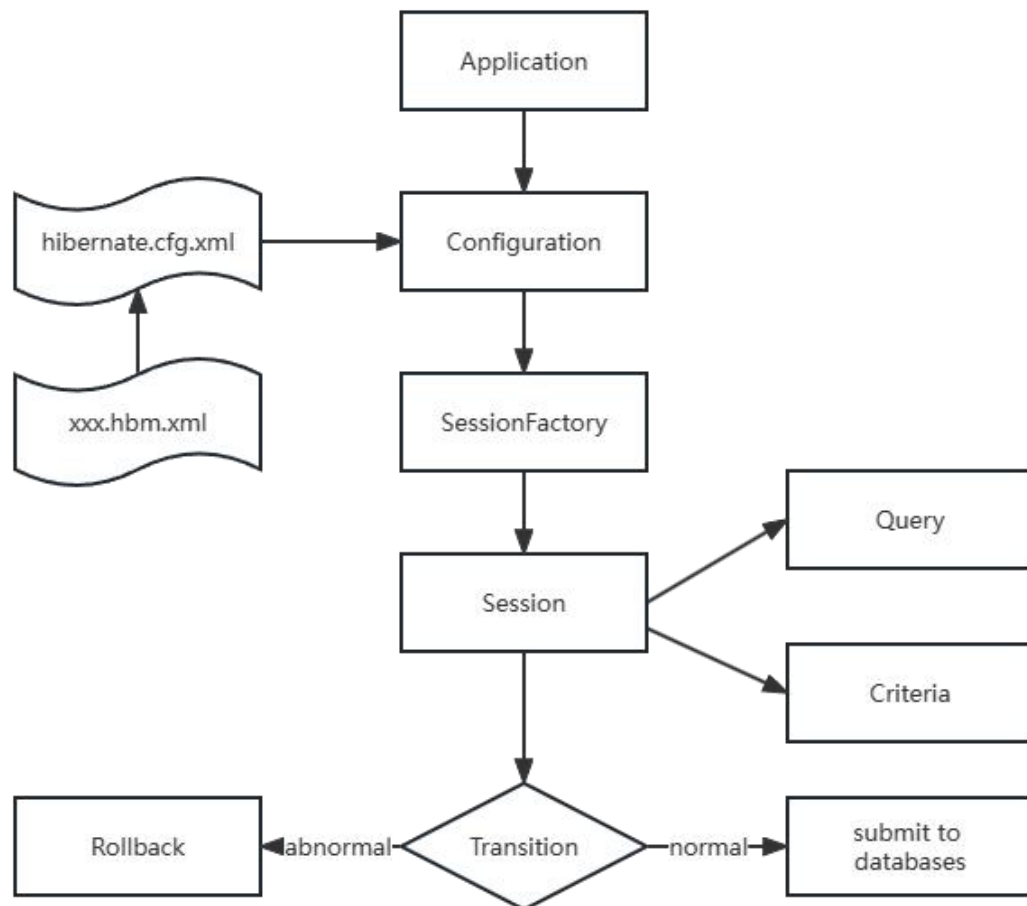
为了实现这个目标，我们在业务层和数据库之间实现了一个持久化层。持久化层是一种中间件，可以自动将对象存储在数据库中，并从数据库加载数据到内存中。通过这样的抽象层，程序将通过会话间接地访问数据库。如下图所示：



我们将选择使用 ORM 中的 Hibernate 框架。原因在于 Hibernate 是一个非常流行的开源 ORM 框架，它提供了一种将 Java 对象映射到数据库表的方式。通过使用 Hibernate，我们可以通过简单的注释或 XML 配置来描述数据模型和映射关系，Hibernate 会自动处理对象与关系数据之间的转换。同时，Hibernate 提供了面向对象的查询语言 HQL，允许使用面向对象的方式进行数据库查询。同时，Hibernate 还支持本地 SQL 查询，使得程序员可以在需要时直接编写原生 SQL 语句。不仅如此，Hibernate 支持多种主流的关系型数据库，包括 MySQL、Oracle、

Microsoft SQL Server 等，并且也支持 NoSQL 数据库 MongoDB。最后，除了自动化 ORM 操作外，Hibernate 还提供了缓存机制，可以有效地减少数据库访问的次数，以提高应用程序的性能。Hibernate 还提供了事务管理、连接池管理等功能，使得开发人员可以更方便地编写数据库相关的代码。

Hibernate 机制如下：



配置(Configuration)对象通常是 Hibernate 应用程序中创建的第一个对象。它通常只在应用程序初始化期间创建一次，表示 Hibernate 所需的配置或属性文件。

Configuration 对象负责管理 Hibernate 运行所需的配置信息，并读取和处理配置文件（例如 hibernate.cfg.xml）中指定的配置属性。使用这些信息，它创建 SessionFactory 类的实例。SessionFactory 是任何 Hibernate 应用程序的关键组件，因为它用于获取 Hibernate Sessions 的实例，这些实例用于执行数据库操作。因此，Configuration 对象在任何 Hibernate 应用程序中都扮演着至关重要的角色，通过提供必要的配置信息来允许 Hibernate 连接到数据库，并创建 SessionFactory，该 SessionFactory 用于管理 Hibernate Sessions 的生命周期。

会话工厂(SessionFactory)是 Hibernate 应用程序中最为关键的对象之一，它负责管理 Hibernate Session 的创建和生命周期，并处理所有与数据库交互的操作。在 Hibernate 中，Session 是执行数据库操作的主要接口。同时，SessionFactory 是一个线程安全的对象，通常只在应用程序启动时创建一次，

然后在整个应用程序运行期间被重复使用。这种设计模式称为单例模式 (Singleton Pattern)。再者, SessionFactory 的创建是一个开销比较大的过程, 因为它需要读取配置文件、扫描实体类等操作。但是, 在 SessionFactory 实例化完成之后, 它会缓存各种映射信息和预编译的 HQL 语句, 从而提高性能。SessionFactory 可以通过 Configuration 对象来创建, 也可以通过 ServiceRegistry 构建器来创建。创建 SessionFactory 时, 需要传递一个 Configuration 对象或者一个包含了 Hibernate 配置信息的 Properties 对象。SessionFactory 则会根据这些配置信息来生成一个或多个与数据库的连接池, 并且管理这些连接池的打开和关闭。最后 SessionFactory 创建好之后, 我们就可以使用它来获取 Session 了。在 Hibernate 中, Session 是一个轻量级的、非线程安全的对象, 它代表了持久化上下文 (Persistence Context), 并提供了对数据库的 CRUD 操作。每个 Session 实例都属于某个 SessionFactory, 它使用 SessionFactory 中提供的连接池进行数据库操作。总之, SessionFactory 是 Hibernate 应用程序的关键组件之一, 它负责管理 Hibernate Session 的创建和生命周期, 并提供了对数据库的连接池、缓存机制等进行管理的功能。对于需要频繁进行数据库操作的应用程序来说, SessionFactory 的优化和使用非常重要。

在 Hibernate 中, Session 是一个重要的概念, 它代表了应用程序与数据库之间的一次会话。通过 Session 对象, 我们可以执行各种数据库操作, 比如插入、更新、删除和查询数据等。同时, Session 也负责管理持久化对象的状态, 确保与数据库的同步。下面是 Session 的一些主要功能:

- 数据库连接管理: Session 封装了 JDBC 的连接, 并负责管理数据库连接的创建、打开、关闭和回收。这样, 我们就无需手动处理底层的数据库连接, 而是可以专注于业务逻辑的实现。
- 对象状态管理: Hibernate 使用对象关系映射 (ORM) 技术, 将 Java 对象映射到数据库表中的记录。Session 负责跟踪持久化对象的状态, 即临时状态、持久状态和游离状态。在不同状态之间转换时, Session 会自动维护对象与数据库记录之间的同步。
- 数据库操作: Session 提供一系列 API, 可以执行各种数据库操作。比如, 我们可以使用 Session 的 `save()` 和 `persist()` 方法来插入新的记录; 使用 `update()` 和 `merge()` 方法来更新已有记录; 使用 `delete()` 方法删除记录; 使用 `get()` 和 `load()` 方法查询记录等。
- 事务管理: Session 还提供了事务管理的功能。我们可以使用 Session 的 `beginTransaction()` 方法开启事务, 然后在事务中执行多个数据库操作。如果所有操作都成功, 则提交事务; 否则回滚事务。
- 查询功能: 除了直接操作数据库外, 我们还可以使用 Session 进行查询操作。Hibernate 提供了多种查询方式, 比如 HQL、Criteria 和 Native SQL 等。我们可以使用这些查询方式来检索数据、进行聚合计算等复杂操作。

Hibernate 中的 Transaction (事务) 用于管理对数据库的多个操作, 从而确保这些操作要么全部成功要么全部失败回滚。在 Hibernate 中, Transaction 对象是通过 Session 对象创建的。一个 Transaction 可以通过 `commit()` 方法来提交, 也可以通过 `rollback()` 方法来回滚。如果不进行提交或回滚, 则 Transaction 将处于挂起状态, 直到 Session 关闭。

Query 是一个接口, 用于执行 HQL 和 SQL 查询, 并返回结果集。它提供了许多方法来设置查询参数、限制结果集大小等功能。在使用 Query 之前, 必须先获

得一个 Session 对象。然后可以通过调用 Session 的 `createQuery()` 方法创建一个 Query 对象。一旦创建了 Query 对象，就可以使用 `setFirstResult()` 和 `setMaxResults()` 方法来限制结果集的大小。`setFirstResult()` 方法指定要从结果集中返回的第一个记录的索引，而 `setMaxResults()` 方法指定最大返回记录数。

在 Hibernate 中，Criteria 是一种用于创建和执行类型安全查询的 API。它通过构建一个包含各种查询条件的条件对象来实现这一点，并将它们应用于 Hibernate 映射类。使用 Criteria API 可以避免手写 SQL 或 HQL 语句，还可以提供更好的可读性和易于维护性，因为它们使用类型安全的方法和属性而不是字符串。

权限管理

为了确保每个用户角色都能顺畅地使用系统中的服务和资源，我们需要实现一个全面、高效且易于使用的用户访问控制机制。这个机制应该能够根据不同角色的权限限制，确保他们只能访问到他们被授权的内容，并且能够快速执行访问控制操作，以提高系统的安全性和可靠性。

我们采用微服务架构并结合常见的权限管理实践，以提供更加细粒度、可靠的用户访问控制机制。这种架构模式具有很多优点。一方面，每个微服务可以通过中心化的授权系统进行权限验证，同时也可以在自己的内部设计更加灵活的权限控制机制。另一方面，在可靠性方面，相比于传统的单一应用模式，基于微服务的系统可以更快地提供服务，并且当某个微服务出现故障时，其他微服务不会受到影响。此外，每个微服务都可以充分利用其硬件资源，而独立扩展某个微服务也不会对其他微服务造成影响。综上所述，微服务架构与权限管理实践的相结合可以有效提高系统的安全性和可靠性，为用户提供更好的使用体验。

在我们的项目中，我们采用 OAuth 2.0 标准协议来进行授权管理。OAuth 2.0 是一种基于 Token 的授权管理系统，相对于基于 cookie 的认证方式，它具有更多优点。例如，基于 Token 的认证机制是无状态的，可以有效防止跨站请求伪造 (CSRF) 攻击，并且支持多站点使用。因此，我们选择使用 OAuth 2.0 协议来进行权限认证，以提高我们系统的安全性和可靠性。

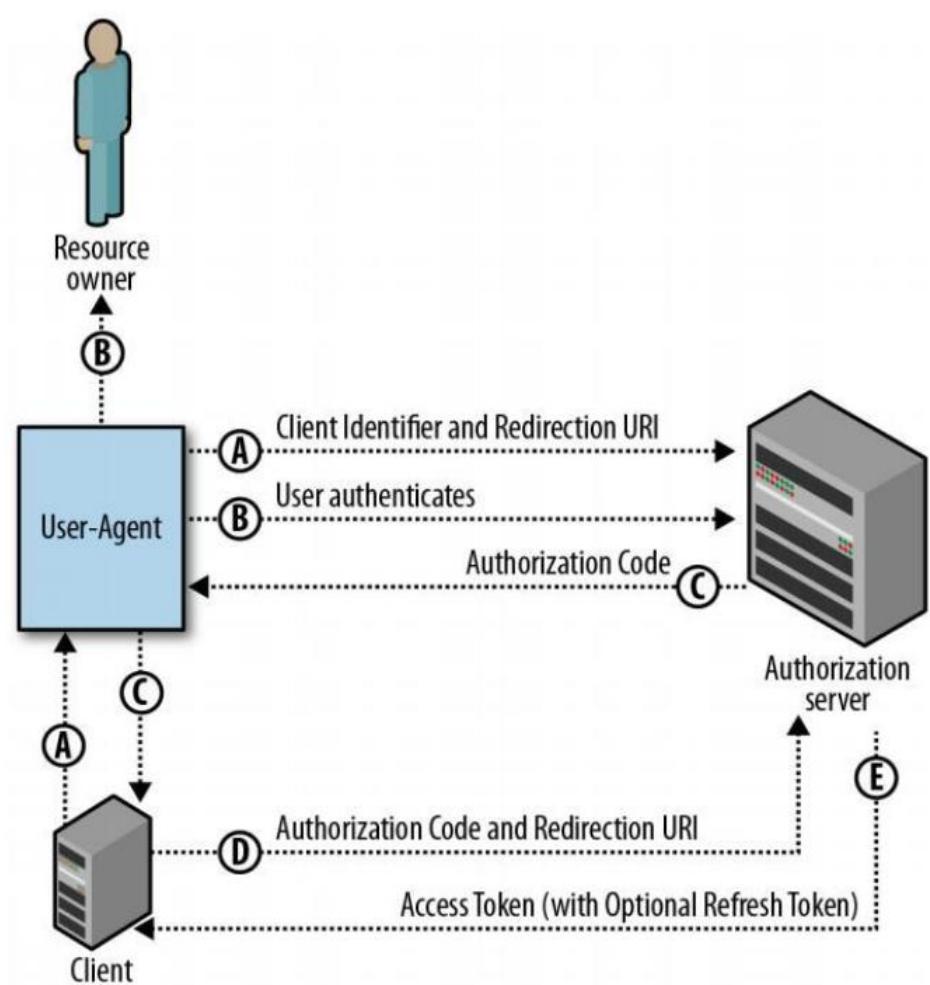
为了在微服务架构中使用 OAuth 2.0，我们采用了常规的 API 网关组件。API 网关是微服务架构的标准组件之一，负责接收来自客户端的请求，并将它们路由到相应的微服务。在这个过程中，API 网关还可以对请求进行鉴权和授权操作，以确保只有经过授权的用户才能访问相应的服务。

具体来说，我们的系统通过引入一个授权服务作为 API 网关的一部分，实现了统一的权限管理机制。当客户端发起请求时，API 网关会将请求发送给授权服务进行身份验证，并获取相应的访问令牌 (Token)。然后，API 网关会将访问令牌附加在请求头中，将请求路由到相应的微服务。在微服务内部，可以通过读取请求头中的访问令牌进行权限校验，以确保只有授权用户才能访问对应的服务。这种方式不仅可以简化权限管理的流程，还可以提高系统运行效率，减少客户端与后端的往返次数。因此，通过引入授权服务和 API 网关的组合方式，我们的系统可以高效、安全地管理用户的权限，并且更好地适应微服务架构。

在我们的实际开发中，我们选择了 Sa-Token 作为授权服务的框架。相比于其他框架，Sa-Token 具有轻量级的特性，使得它在微服务架构中具有更好的灵活性和适应性，便于我们将其整合到整个系统中。同时，Sa-Token 还提供了 Sa-OAuth2 模块，基于 RFC-6749 标准编写，可以非常方便地实现 OAuth 2.0 授

权认证。除此之外，Sa-Token 还支持常见的二级认证，这在我们的项目中也是较为常见的需求。

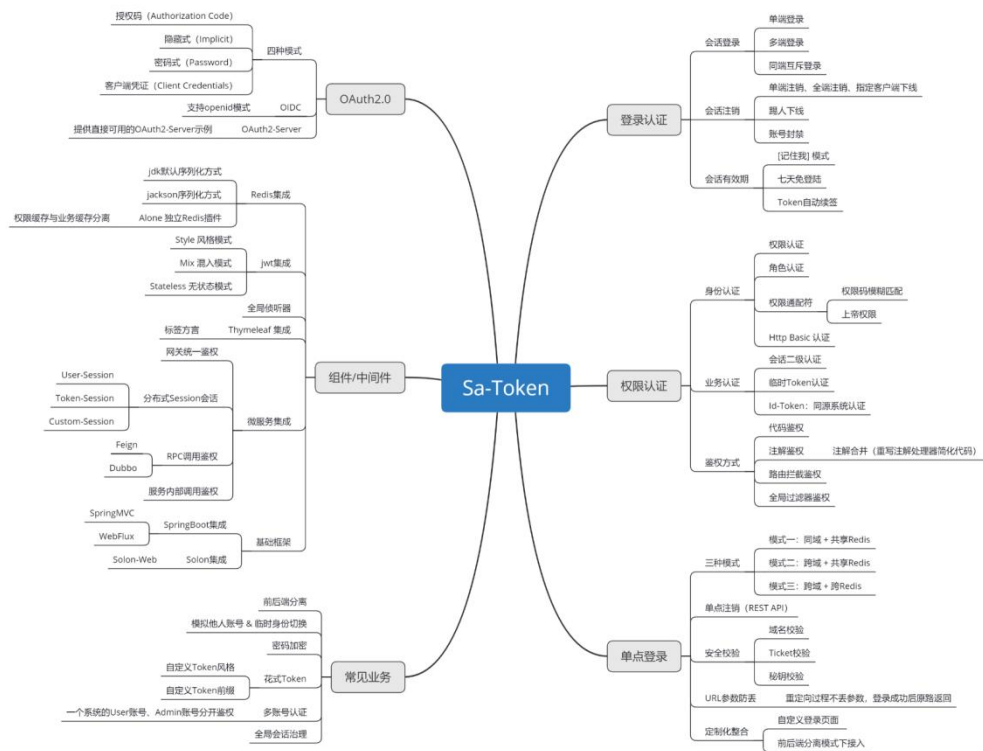
OAuth 2.0 授权过程如下图所示：



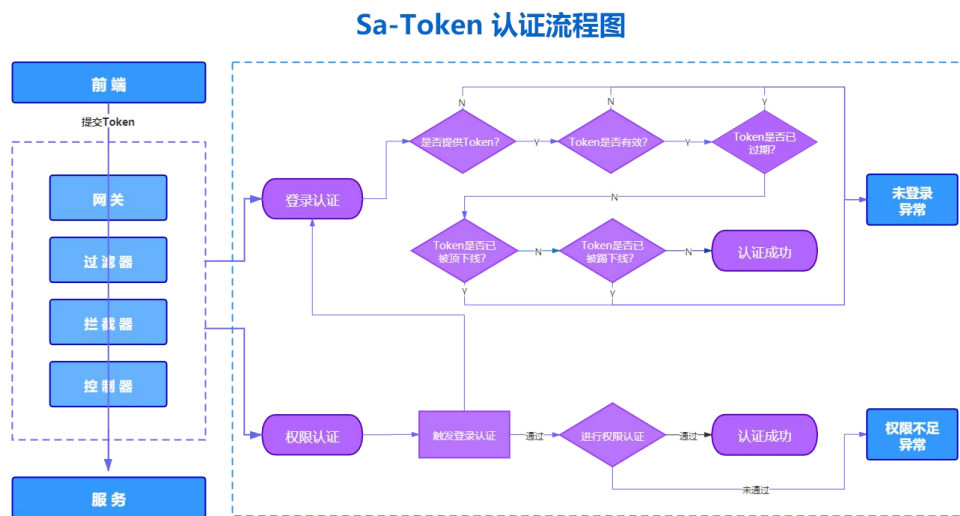
典型的 Sa-Token 认证流程如下：首先，客户端向 API 网关发送请求，并附带访问令牌(Token)。API 网关收到请求后，会将 Token 发送给 Sa-Token 进行验证。Sa-Token 会对 Token 进行解析，并根据解析结果判断请求是否有效。如果 Token 验证通过，则 API 网关会将请求路由到相应的微服务中进行处理；否则，Sa-Token 将拒绝该请求并返回错误信息。

在 Sa-Token 中，Token 的生成、解析和管理等操作都由框架自动完成，无需手动编写代码。同时，Sa-Token 还提供了丰富的权限控制功能，例如角色、权限、登录设备等，可以满足复杂的权限管理需求。总之，我们选择 Sa-Token 作为授权服务框架，可以更加高效、安全地管理用户的权限，并提供灵活的二级认证机制，为我们的系统开发带来了很大的便利。

Sa-Token 功能如下所示：



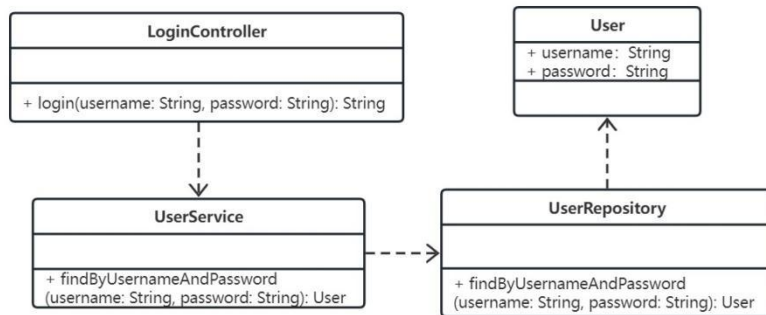
Sa-Token 流程如下图所示：



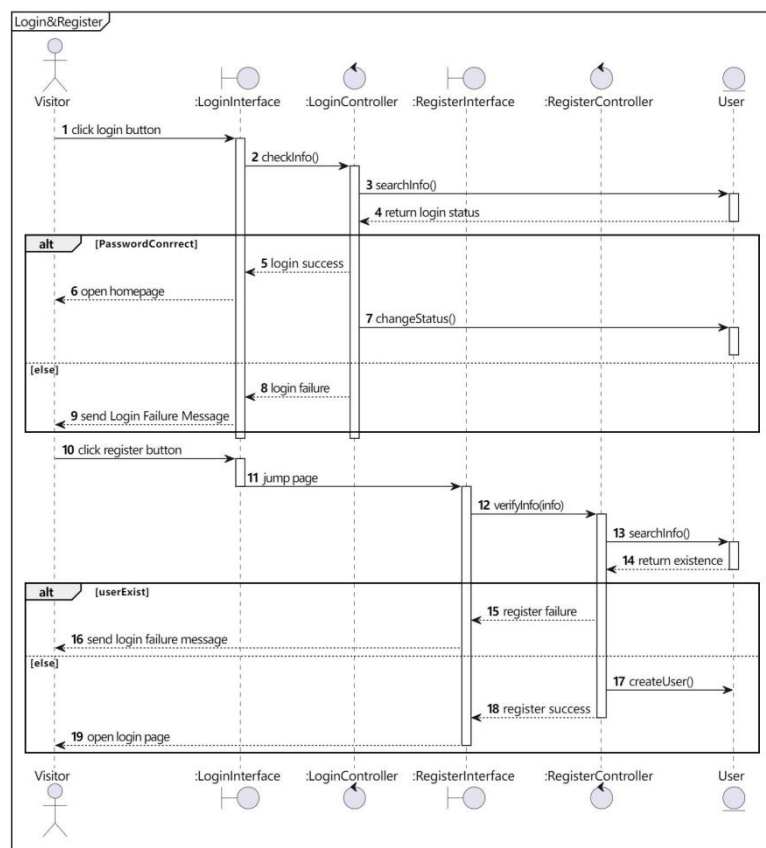
四、用例实现

用户登录

类图与交互图如下：



交互图



实现步骤如下

1. 用户在登录表单中输入用户名和密码。
2. 登录界面会先对信息进行输入级别的检查，即系统会拒绝格式不合法的登录信息（如空用户名和空密码）。
3. 之后，信息将被解析为 JSON 格式。登录请求将被发送到 Spring MVC。
4. 同时，带有登录信息的 JSON 文件将通过 REST API 发送到数据库，同时向数据库发送相应的查询请求。
5. 数据库中的用户表存储了用户的所有信息。数据库查询用户表，查找给定信息是否在用户表中。如果存在，则将成功消息返回给用户管理子系统，即可以在数据库中找到对应的用户名和密码。

6. 在得到查询结果后，我们的系统也会启动一个验证过程来检查用户帐户是否被允许登录。身份验证过程将由 **Sa-Token** 处理。通过调用 **StpUtil.getUserByToken** 方法，会返回用户的 **token** 是否合法。
7. 如果证明 **token** 是合法的（如果出现用户封禁等情况，其 **token** 会被设置为非法），会生成一个新的登录 **token**，发送到登录子系统，并保存到本地。
8. 每当用户在我们系统中切换网页时，**Sa-Token**（通过 **REST API**）将在登录状态级别（操作由 **StpUtil.getLoginIdByToken** 方法提供支持）检查帐户的 **token**，以防系统崩溃或其他错误情况。具体来说，当用户的登录状态受到系统故障影响时，**token** 将被禁用，系统将拒绝用户继续使用该服务。

下面的代码展示了如何在 **Spring MVC** 中处理登录请求，并使用 **Sa-Token** 进行身份验证。

```
// LoginController.java
public class LoginController {
    private UserService userService;
    public String login(String username, String password) {
        User user = userService.findByUsernameAndPassword(username, password);
        if (user != null) {
            // 登录成功，使用Sa-Token进行身份验证
            loginWithSaToken(user.getId());
            return "redirect to home page";
        } else {
            // 登录失败
            return "redirect to login page";
        }
    }
}

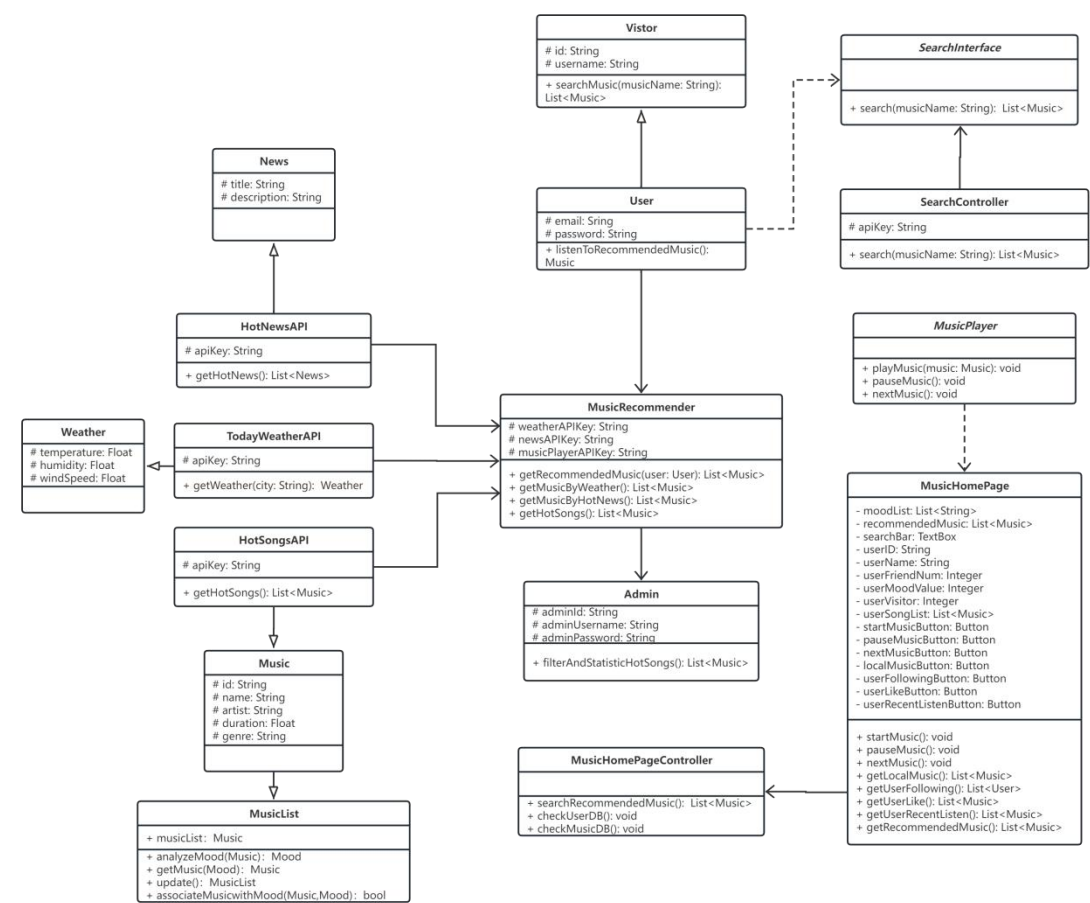
// UserService.java
public class UserService {
    private UserRepository userRepository;
    public User findByUsernameAndPassword(String username, String password) {
        // 查询数据库，查找用户名和密码匹配的用户
        return userRepository.findByUsernameAndPassword(username, password);
    }
}

// UserRepository.java
public interface UserRepository {
    User findByUsernameAndPassword(String username, String password);
}
```

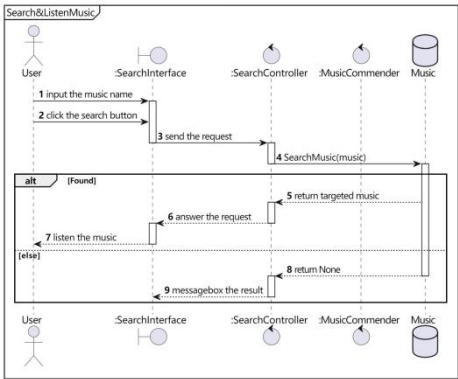
在这个示例中，我们首先在 **LoginController** 中定义了一个 **login** 方法，这个方法接收用户名和密码作为参数。然后，我们调用 **UserService** 的 **findByUsernameAndPassword** 方法，这个方法会查询数据库，查找用户名和密码匹配的用户。如果找到了匹配的用户，我们就使用 **Sa-Token** 的 **login** 方法进行身份验证，然后重定向到主页。如果没有找到匹配的用户，我们就重定向到登录页。

音乐推荐

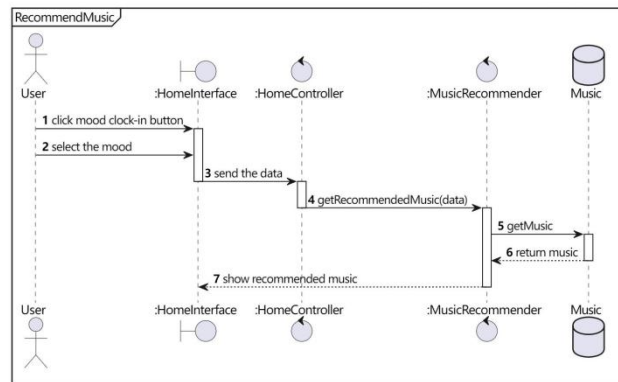
其类图与交互图如下：



搜索&听音乐



音乐推荐至个人



实现步骤如下：

1. 用户在搜索界面输入音乐名称并点击搜索按钮。这个过程涉及到前端的 **JavaScript** 和 **HTML** 技术，使用到的库为 **Vue.js** 来构建用户界面。
2. 搜索界面将请求发送到搜索控制器。这个过程涉及到 **Spring MVC** 的控制器和视图模型，使用 **HTTP** 协议传输用户的请求。
3. 搜索控制器在音乐数据库中搜索音乐。这个过程涉及到 **REST API** 的使用，通过 **REST API**，我们可以从网易云的 **API** 获取音乐数据，可能使用到的技术包括 **JSON** 数据格式和 **HTTP** 请求方法。
4. 如果找到了目标音乐，音乐数据库将目标音乐返回给搜索控制器，搜索控制器回答请求，搜索界面让用户听音乐。这个过程涉及到 **Spring MVC** 的数据绑定和视图渲染，可能使用到的技术包括 **Thymeleaf** 或 **JSP**。
5. 如果没有找到目标音乐，音乐数据库返回 **None**，搜索控制器在搜索界面显示结果消息框。这个过程涉及到 **Spring MVC** 的异常处理和消息提示，可能使用到的技术包括 **Spring** 的错误处理机制和 **Bootstrap** 的消息提示组件。
6. 用户在主页界面点击心情打卡按钮并选择心情。这个过程涉及到前端的 **JavaScript** 和 **HTML** 技术，使用到的库为 **Vue3** 来构建用户界面。
7. 主页界面将数据发送到主页控制器。这个过程涉及到 **Spring MVC** 的控制器和视图模型，使用 **HTTP** 协议传输用户的请求。
8. 主页控制器获取音乐推荐器推荐的音乐。这个过程涉及到机器学习算法，用于根据用户的心情推荐音乐，可能使用到的技术包括 **Python** 的 **Scikit-learn** 库或 **TensorFlow** 库。
9. 音乐推荐器从音乐数据库获取音乐，音乐数据库返回音乐给音乐推荐器。这个过程涉及到 **REST API** 的使用，通过 **REST API**，我们可以从网易云的 **API** 获取音乐数据，可能使用到的技术包括 **JSON** 数据格式和 **HTTP** 请求方法。
10. 音乐推荐器在主页界面显示推荐的音乐。这个过程涉及到 **Spring MVC** 的数据绑定和视图渲染，可能使用到的技术包括 **Thymeleaf** 或 **JSP**。

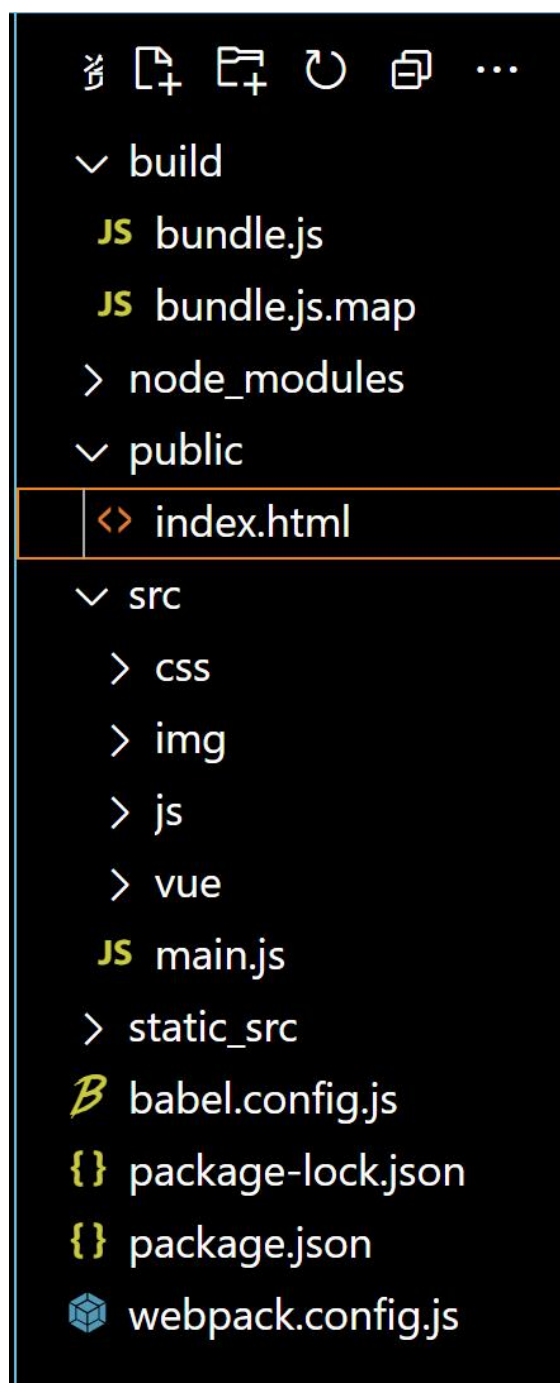
五、产品原型进展

5.1 前端原型

在本项目中，我们使用 VUE3 框架开发前端，从而进行 HW 的网页设计。同时，我们还使用了 npm 作为前端项目包管理工具，并使用 webpack 进行打包。

以登录界面为例

文件目录如下：

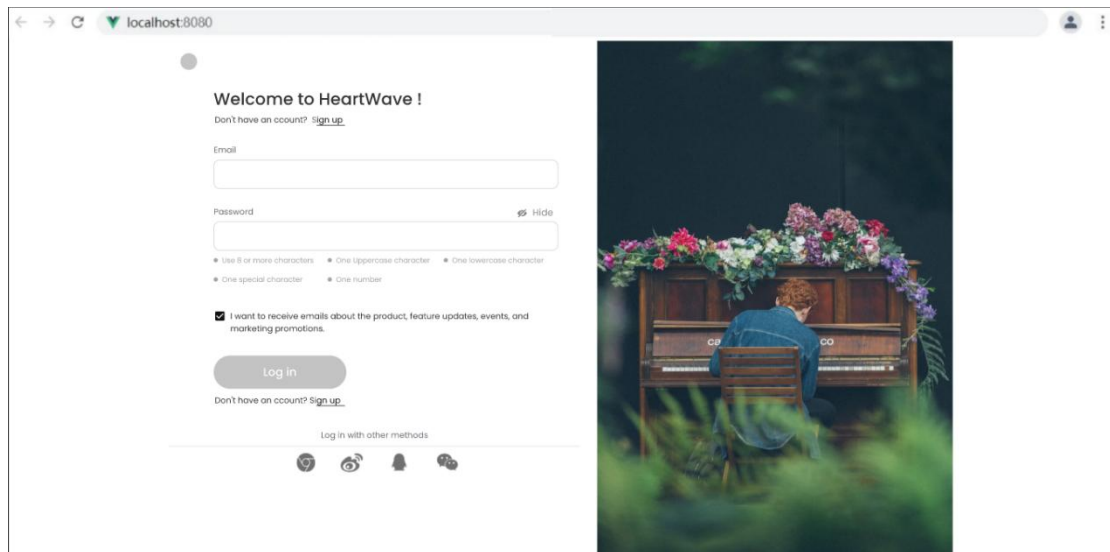


我们在终端中输入`npm run serve`指令，便可以在 localhost 为 8080 的端口访问到我们的页面。

```
<i> [webpack-dev-server] Project is running at:  
<i> [webpack-dev-server] Loopback: http://localhost:8080/
```

```
webpack 5.75.0 compiled successfully in 5043 ms  
asset bundle.js 99 bytes [emitted] (name: main)
```

我们的登录页面如下：



后端原型

在本项目中，我们使用 Java 进行后端的开发。为了将前后端连接起来，我们使用相应的 API 保证两个部分的通信。

以下，我们将以获得用户情绪数据为例进行展示。该请求为 GET 请求，REST API 为 /api/emotion-analysis/history，参数为请求返回心情历史记录的用户 id。

首先，我们需要使用 Java 语言建立一个相关用户情绪数据的类以及存储该类和通过 id 寻找到某个实例的方法，如下所示：

```
@Data  
@Entity  
public class UserMoodData {  
    @Id  
    private String userid;  
    private Date date;  
    private int moodcount;  
    private Map<String, Integer> mood;  
}  
  
@Repository  
public interface UserMoodDataRepository extends JpaRepository<UserMoodData, String> {  
    UserMoodData findOneById(String id);  
}
```

```
    Iterable<UserMoodData> mostRecent(int t):  
}
```

因此，我们可以确定该功能相应的 API 函数

首先，我们初步定义一些常见的 HTTP 状态码，帮助我们更好地获得 API 请求结果的状态。

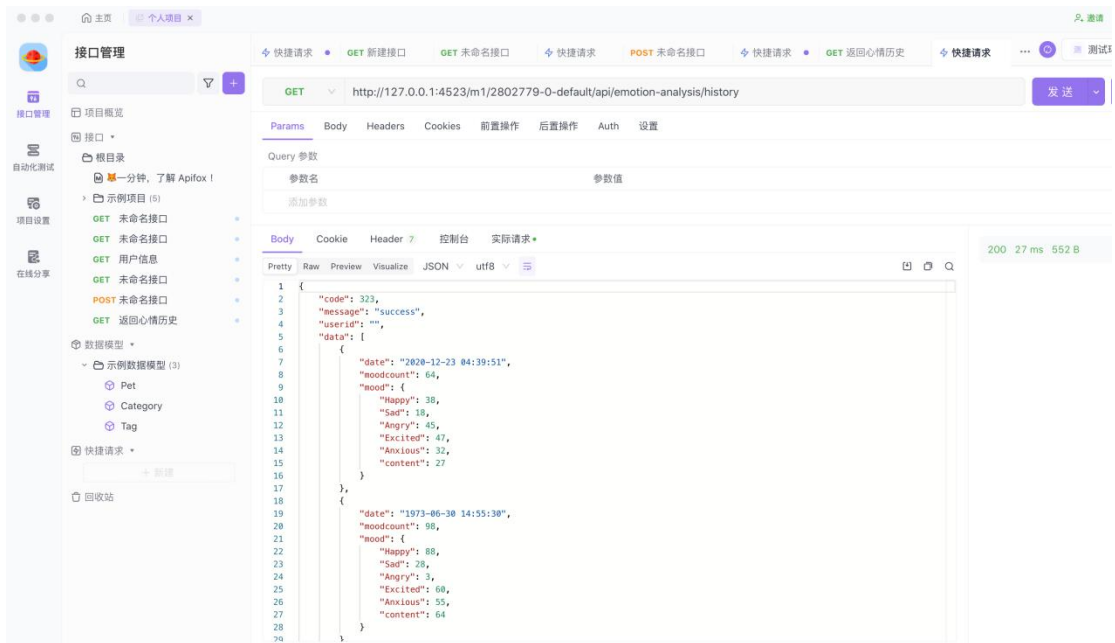
如下所示：

```
public enum HttpStatus {  
    OK(200, "OK"),  
    CREATED(201, "Created"),  
    ACCEPTED(202, "Accepted"),  
    NO_CONTENT(204, "No Content"),  
    BAD_REQUEST(400, "Bad Request"),  
    UNAUTHORIZED(401, "Unauthorized"),  
    FORBIDDEN(403, "Forbidden"),  
    NOT_FOUND(404, "Not Found"),  
    METHOD_NOT_ALLOWED(405, "Method Not Allowed"),  
    INTERNAL_SERVER_ERROR(500, "Internal Server Error"),  
    SERVICE_UNAVAILABLE(503, "Service Unavailable");  
}
```

然后，我们通过用户 id 获得情绪数据的 API 函数如下：

```
@RestController  
@RequestMapping("/api/emotion-analysis")  
public class UserMoodDataController {  
  
    @Autowired  
    private UserMoodDataRepository userMoodDataRepository;  
  
    @GetMapping("/history/{id}")  
    public ResponseEntity<?> getUserMoodDataById(@PathVariable String userid) {  
        try {  
            UserMoodData moodData = userMoodDataRepository.findOneById(userid);  
            if (moodData != null) {  
                return new ResponseEntity<>(moodData, HttpStatus.OK);  
            } else {  
                return new ResponseEntity<>("User mood data not found",  
HttpStatus.NOT_FOUND);  
            }  
        }  
        catch (Exception e) {  
            return new ResponseEntity<>("Error retrieving emotion data",  
HttpStatus.INTERNAL_SERVER_ERROR);  
        }  
    }  
}
```


在 Apifox 中测试如下：



六、自我反馈

苏家铭：

经过一个学期的学习，我深刻地明白了一个项目的前期设计基础流程，从一个 idea 的出现，到最后产品发布与运维，每一个环节都至关重要。从前的自己连一个项目需要画的用例建模都不知道，到最后的明白用例图、关系图、类图等一系列重要的模型，自己对于一个项目有了整体上宏观的把握，也能在微观上理解项目开发的步骤。

一个项目的系统分析与设计从来不是一个人的事情，而是一个团队的共同努力的造就。所以我也很感谢我的队友在这和一个学期的团结协作，相互进步，我们在一起学习新知识，相互扶持与帮助，积累了深厚的友谊，本人的团队协作能力也得到了质的飞跃。

杨滕超：

通过学习系统分析与设计，我充分地了解到原来一个软件从开发到最终成型并不是直接上去堆代码，而是需要前面设计与分析的铺垫。从用户需求到 UML 图、原型界面，再到函数设计和 API 设计，最后才是写相应功能对应的函数。终于我发现，最初的设计其实很大的影响到后面的实现。

同时，能和本组的几位优秀的队友合作是我一生的荣幸，这个过程不仅锻炼了团队合作能力，并且从中积累了丰富的合作经验。大家都积极沟通，工作认真诚恳，为我树立了良好的榜样。

王蔚达：

这个学期的系统分析与设计课程带我走过了系统从构建到实现的全程，使我对系统开发有了全面理解。我学会了通过用例图和用例说明书清晰地表达系统用

例，利用活动图和顺序图详细描述用例功能。这些设计工具提升了我对系统设计的理解，并为我提供了明确的视觉参考。

我还深入了解了系统设计过程中可能涉及的机制，以至于对未来的项目开发有了更清晰的指导。现在我知道如何规范开发流程，避免初学者错误，不再对复杂的系统设计感到无从下手。

同时，我在团队协作和沟通上得到了锻炼。通过与团队成员协作，我提高了团队合作能力，理解了在团队中如何发挥作用，与他人合作，找到自己的位置。这些宝贵的经验将对我未来的学习和工作有巨大帮助。

简而言之，这门课程不仅让我掌握了专业知识和技能，也增进了我对团队协作的理解。我已做好准备，期待将所学应用于实践，迎接未来的挑战。

莫益萌：

系统分析与设计是一门十分实用，也十分难忘的课程。在对我们的整个项目进行了敏捷分析和架构分析的过程中，我学到了很多需求分析和架构设计的知识。我学会了如何寻找目标用户，如何绘制 UML 图，如何对整个系统应该采用的架构进行分析。在课程中，通过与队友的讨论与协作，我获得了一定的团队协作的能力，也得到了很多来自队友的全新的想法。总而言之，系统分析与设计这门课，能够全面提升我们软件工程专业学生的团队协作能力和工程能力，通过修读这门课，我们受益颇多。

储岱泽：

通过系统分析与设计课程的学习，在理论与实践的洗礼下，我获得了宝贵的知识和经验，掌握了设计系统所需的技能和工具，受益匪浅。课程让我从低级的思维方式中解放出来，拥有了更高层次的视角和规划能力。我学会了使用 UML 图、原型图等工具来规范和描述系统，学会了使用各类设计平台进行精美的 UI 设计，深入分析功能和架构。同时，通过这门课我也意识到了系统分析与设计的重要性，它是确保软件开发顺利进行的关键。通过团队合作项目，我锻炼了沟通协作能力，扩展了技术栈。这门课程不仅为我成长为一名优秀的软件工程师打下基础，更让我对编程的意义有了深刻的理解。我将继续努力将所学应用到实践中，不断提升自己的能力，为改善人们的生活贡献更多价值。

七、成员贡献

在完成这一项目的过程中，所有成员都在积极讨论，认真参与，团队成员根据自己的兴趣和能力进行了相应的分工，按时按量完成自己的任务。整个过程相处和谐，沟通效率高。

以下是各个成员的任务分配情况：

Contributor	Introduction	构架完善	设计机制	用例实现	产品原型
苏家铭	√				√
杨滕超			√		√
王蔚达			√	√	
莫益萌		√	√		

储岱泽	√	√			
-----	---	---	--	--	--

成员名单及得分占比：

苏家铭	100%
杨滕超	100%
王蔚达	100%
莫益萌	100%
储岱泽	100%