VibeVault 项目评分报告

学号: N/A

姓名: N/A

班级: N/A

Commit: d9cbab1

成绩汇总

项目	得分
编程测试	60/60
报告	10/10
前端	10/10
总分	**80/80**

后端开发反思报告

---

1. 我遇到的最大挑战

在开发过程中，我遇到的最大挑战是 **JPA 实体类的双向关联维护**问题。

起初，我在实现 `Playlist` 和 `Song` 的一对多关系时，只是简单地在 `Playlist` 类中添加了 `@OneToMany` 注解，以为这样就完成了。但当我运行测试时，发现添加歌曲到歌单后，歌曲的 `playlist` 字段始终是 `null`，导致测试失败。

我尝试了多种方法排查：首先检查了注解是否正确，发现 `mappedBy` 属性设置没问题；然后在数据库控制台查看数据，发现 `songs` 表中的 `playlist_id` 确实是空的。这让我意识到问题出在双向关联的维护上。

经过查阅 JPA 文档，我终于明白了：在双向关联中，**关系的维护方是"多"的一方**（即 `Song`），而 `Playlist` 只是被动方。因此，仅仅调用 `songs.add(song)` 是不够的，还必须同时调用 `song.setPlaylist(this)` 来维护另一端的引用。最终的实现是：

```
public void addSong(Song song) {
    songs.add(song);
    song.setPlaylist(this);  // 关键：维护双向关联
}
```

这个问题让我深刻理解了 **ORM 框架只是工具，底层的关系型数据库原理仍然需要掌握**。双向关联看似方便，但如果不理解其工作机制，反而会成为 bug 的温床。

---

2. 如果重新做一遍

回顾这次开发，有几个地方我觉得可以做得更好：

首先是异常处理的设计。目前 `ResourceNotFoundException` 和 `UnauthorizedException` 都继承自 `RuntimeException`，虽然能用，但错误信息的格式不够统一。如果重新做，我会设计一个通用的 `ApiException` 基类，包含错误码、错误信息和详细描述，这样前端解析错误会更方便。

其次是 DTO 转换的方式。目前我在 `PlaylistServiceImpl` 中用私有方法 `toDTO()` 进行转换，虽然能工作，但随着实体增多，这种方式会导致 Service 类越来越臃肿。更好的做法是使用 MapStruct 这样的映射框架，或者至少把转换逻辑抽取到独立的 Mapper 类中。

第三是安全配置的细化。目前的 `SecurityConfig` 把所有路径规则都写在一起，如果后续 API 增多，这个配置会变得很长且难以维护。我会考虑使用 `@PreAuthorize` 注解将权限控制下放到 Controller 方法级别，这样更直观，也更容易测试。

最后是测试覆盖。虽然公开测试都通过了，但我没有自己编写额外的单元测试来覆盖边界情况。如果时间充裕，我会为 Service 层编写更多测试用例，特别是针对权限检查和异常场景的测试。

3. AI 协同开发经验

这次开发中，AI 在以下场景帮助很大：

场景一：JWT 实现。我对 JWT 的 API 不太熟悉，于是问 AI："如何使用 jjwt 库生成和验证 JWT token？"AI 给出了完整的代码示例，包括使用 `Jwts.builder()` 生成 token 和 `Jwts.parser()` 解析 token。这个回答直接帮我节省了大量查文档的时间。不过 AI 给的示例用的是旧版 API（`setSubject()` 而非 `subject()`），我需要根据实际使用的 jjwt 0.12.x 版本进行调整。这让我学到：**AI 的知识可能有时效性，需要验证是否与当前使用的库版本匹配**。

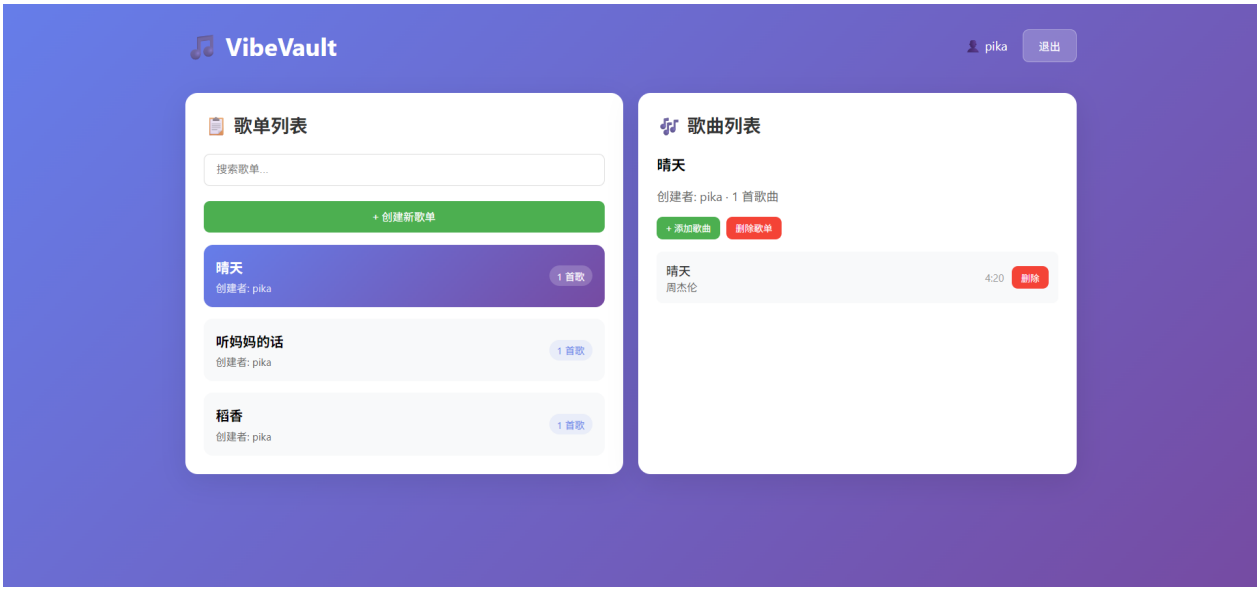
场景二：Spring Security 配置。我问 AI："如何配置 Spring Security 让某些路径公开访问，其他需要 JWT 认证？"AI 给出了 `SecurityFilterChain` 的配置方式，这个回答很有用。但 AI 最初建议我返回 403 而非 401，我查阅 HTTP 规范后发现，未认证应该返回 401，而无权限才是 403。这让我意识到：**不能盲目接受 AI 的回答，特别是涉及规范和最佳实践的问题，需要自己判断**。

总结：AI 是很好的"快速入门"工具，能帮我快速了解一个陌生技术的基本用法。但它的回答可能过时、可能有细节错误、可能不符合最佳实践。**正确的使用方式是把 AI 当作"高级搜索引擎"，而非"权威答案"**。我需要理解 AI 给的每一行代码，而不是复制粘贴后祈祷它能工作。

前端开发反思报告

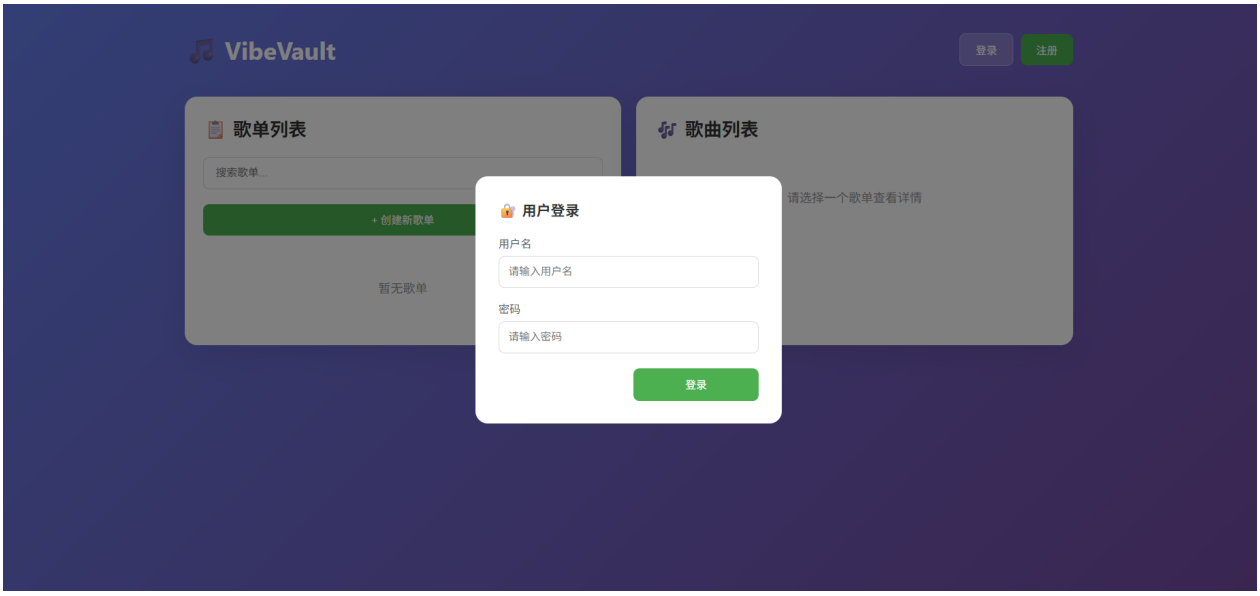
1. 我的界面展示

1.1 首页 - 歌单列表



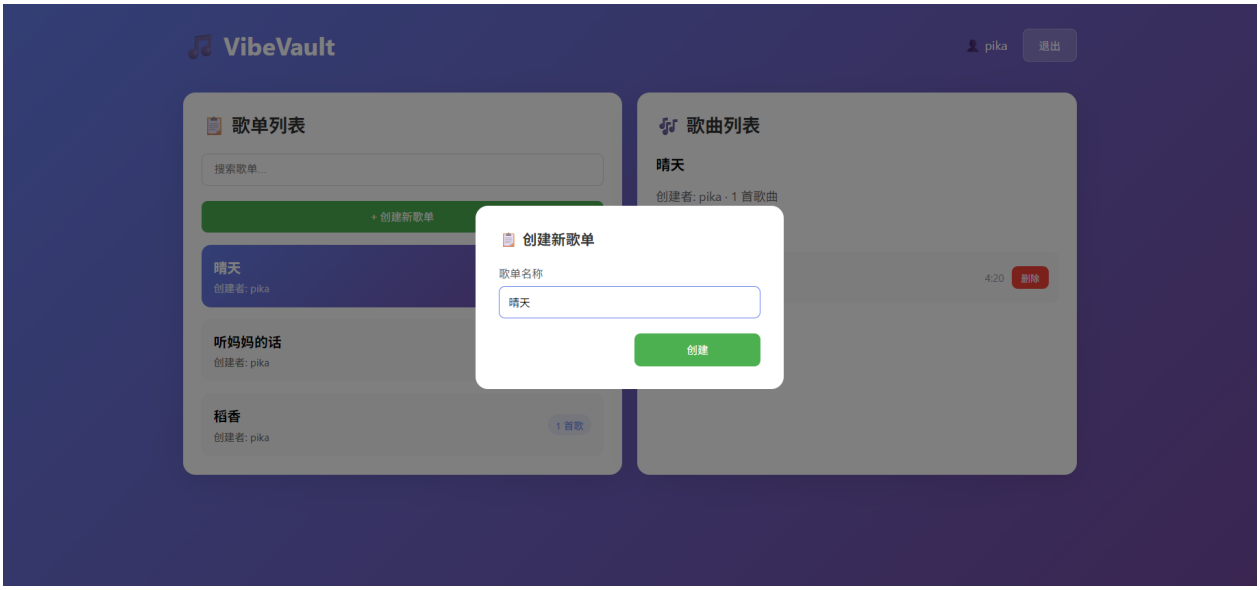
这是 VibeVault 的首页，采用了紫色渐变背景营造音乐氛围。左侧卡片展示所有歌单列表，每个歌单显示名称、创建者和歌曲数量。顶部搜索框支持实时搜索歌单。右上角显示登录/注册按钮，方便用户快速进入系统。

1.2 用户登录



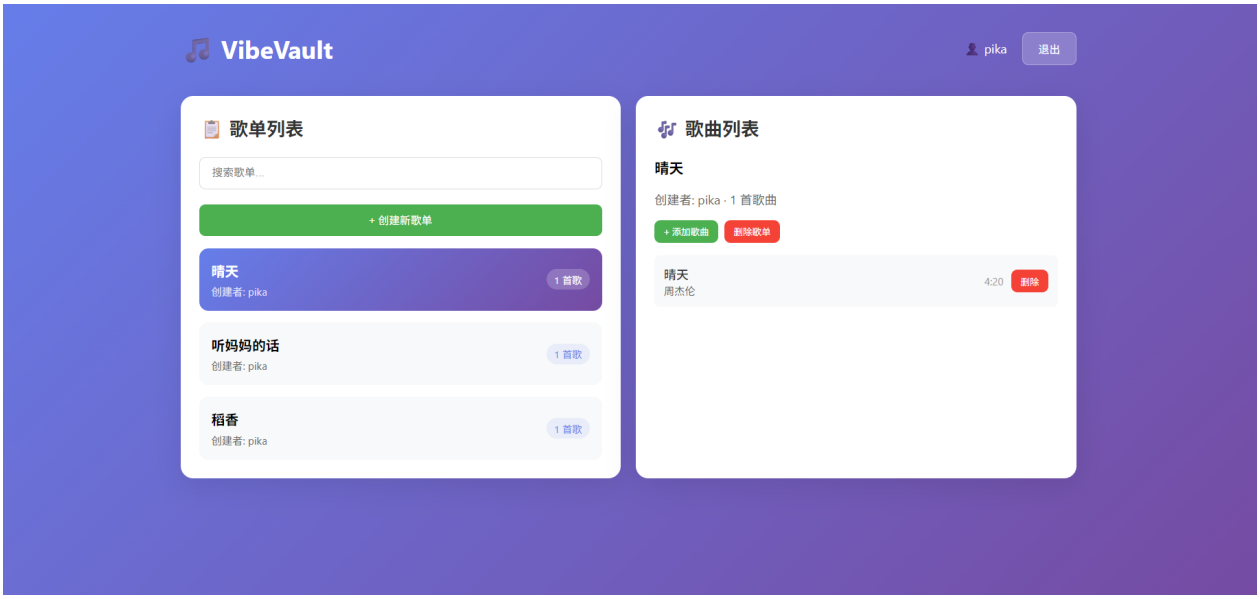
点击"登录"按钮弹出登录窗口，采用模态框设计避免页面跳转。表单包含用户名和密码输入框，底部有取消和登录按钮。输入框有 focus 状态高亮，提升用户体验。

1.3 创建歌单



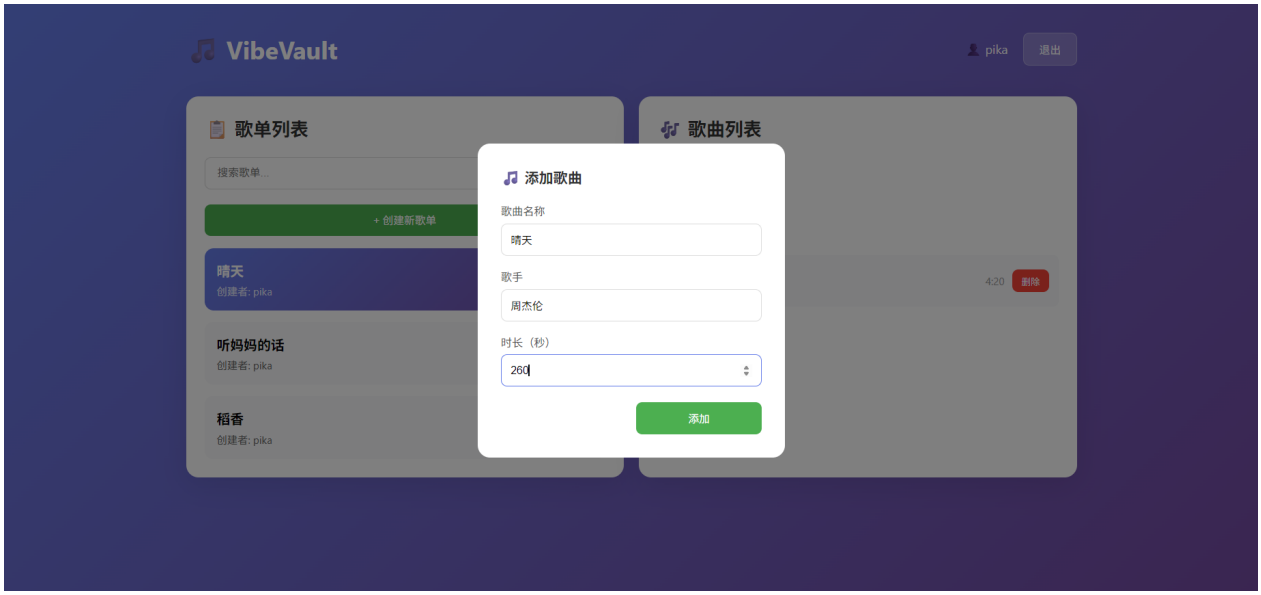
登录后可以创建新歌单。点击"创建新歌单"按钮弹出创建窗口，输入歌单名称即可创建。创建成功后列表自动刷新，新歌单会立即显示在列表中。

1.4 歌单详情与歌曲管理



点击左侧歌单，右侧显示详情。包括歌单名称、创建者、歌曲数量等信息。如果是自己的歌单，会显示"添加歌曲"和"删除歌单"按钮。歌曲列表展示每首歌的名称、歌手和时长，支持单独删除。

1.5 添加歌曲



在自己的歌单中点击"添加歌曲"，弹出添加窗口。需要填写歌曲名称、歌手和时长（秒）。添加成功后歌单详情自动刷新，新歌曲立即显示。

---

2. 我遇到的最大挑战

前端开发中最大的挑战是 **JWT Token 的管理和 API 认证对接**。

最初我直接在每个需要认证的请求中手动添加 `Authorization` 头，代码写得很乱。更麻烦的是，登录成功后 Token 没有持久化，刷新页面就丢失了，用户需要重新登录。

我的解决过程： 1. 首先把 Token 存储到 `localStorage`，这样刷新页面后仍能保持登录状态 2. 封装了统一的请求头处理，在每个需要认证的 `fetch` 调用中自动添加 `Authorization: Bearer ${token}` 3. 在页面加载时检查 `localStorage` 中是否有 Token，如果有则自动恢复登录状态

另一个挑战是**跨域问题**。开发时前端和后端端口不同，导致请求被浏览器拦截。最终我选择把前端 HTML 放在 Spring Boot 的 `static` 目录下，这样前后端同源，彻底避免了跨域问题。

这个过程让我理解了：**前后端分离架构中，认证状态的管理是核心问题**。Token 的存储、传递、过期处理都需要仔细设计。

---

3. 如果重新做一遍

如果重新做，我会改进以下几点：

首先是使用前端框架。目前用原生 JavaScript 写的代码，DOM 操作和状态管理都很原始。如果用 Vue 或 React，可以更优雅地处理数据绑定和组件复用，代码也更容易维护。

其次是增加加载状态。目前点击按钮后没有 loading 提示，用户不知道请求是否在进行中。我会添加 loading 动画，在请求期间禁用按钮，避免重复提交。

第三是完善错误处理。目前只是简单地用 Toast 提示"操作失败"，没有具体说明原因。我会解析后端返回的错误信息，给用户更明确的提示，比如"用户名已存在"而不是"注册失败"。

最后是响应式设计。虽然加了媒体查询，但在手机上的体验还不够好。我会使用更完善的响应式方案，确保移动端也有良好的使用体验。