



# 高级程序设计：自由命题的 Web 应用 实验报告

姓名 杨睿骐

学号 3230104433

实验日期 November 8, 2025

指导老师 耿晨歌

# Contents

<b>1 实验目的</b>	<b>1</b>
<b>2 项目详述</b>	<b>1</b>
<b>3 功能演示与技术综述</b>	<b>1</b>
3.1 一键获取歌曲推荐 . . . . .	3
3.2 旅途回忆可视化与编辑 . . . . .	4
3.3 旅途回忆检索 . . . . .	5
3.4 个性化偏好定制 . . . . .	5
<b>4 主要技术详述与核心代码实现</b>	<b>6</b>
4.1 前端 React + Next.js 技术 . . . . .	6
4.2 智能体技术与推荐算法设计 . . . . .	6
4.2.1 智能体技术: CLIP 图像情绪分析 . . . . .	6
4.2.2 推荐算法: 余弦相似度 . . . . .	7
4.3 后端 FastAPI 技术 . . . . .	9
4.4 数据库存储技术 . . . . .	10
4.4.1 Create (增) - createDiary . . . . .	10
4.4.2 Read (查) - getDiaryById . . . . .	13
4.4.3 Update (改) - updateDiary . . . . .	14
4.4.4 Delete (删) - deleteDiary . . . . .	16
<b>5 总结与展望</b>	<b>17</b>

## 1 实验目的

- 使用 Java/Python 制作 Web 应用
- 用 API 形式建立后端，即需要建立一个客户端程序，语言自选（java, js, python, shell）
- 使用数据库存储数据：有增、删、改、查的操作

## 2 项目详述

**PicMusic** 是首个将旅行日记与音乐相结合的应用，主要贡献包括：

- 它解决了传统旅行日记“只有照片和文字，缺乏氛围感”的问题，将听感融入记忆，促进旅行记忆多模态管理的发展。
- 它解决了传统歌曲推荐陷入协同过滤算法的瓶颈，通过“辨图识歌”，使得用户获取更加个性化的场景音乐，用于聆听，也可用于视频配乐。

**PicMusic** 的核心功能是：当用户在撰写旅行日记并上传照片时，智能体会“端详”这张照片，并自动分析出它所蕴含的情绪——比如，一张日落海滩的照片可能是“平静”和“浪漫”的，而一张登山顶峰的照片可能是“激动”和“快乐”的。随后，系统会利用这种情绪，去庞大的音乐库（tracks 表）中，为用户挑选出情绪最匹配的几首歌曲。最终，当用户回顾这段旅途时，不仅能看到照片和文字，还能听到当时那首“情绪 BGM”，从而实现一种“有声的回忆”。

预览功能演示视频已发布在项目主页 (<https://picmusicweb.github.io/>)，代码现已开源，发布在 ([https://github.com/bucolic-ruiqi/PicMusic\\_web](https://github.com/bucolic-ruiqi/PicMusic_web))

## 3 功能演示与技术综述

**PicMusic** 的所有功能均建立在 Next.js 前端、FastAPI 后端和 MySQL 数据库的三层架构之上，核心架构与主要技术流如图 1 和 2。PicMusic 页面设计简洁，以紫色为主色调，易引起情绪共鸣，同时兼顾深色模式，首页如图 3 所示。



Figure 1: 核心技术流

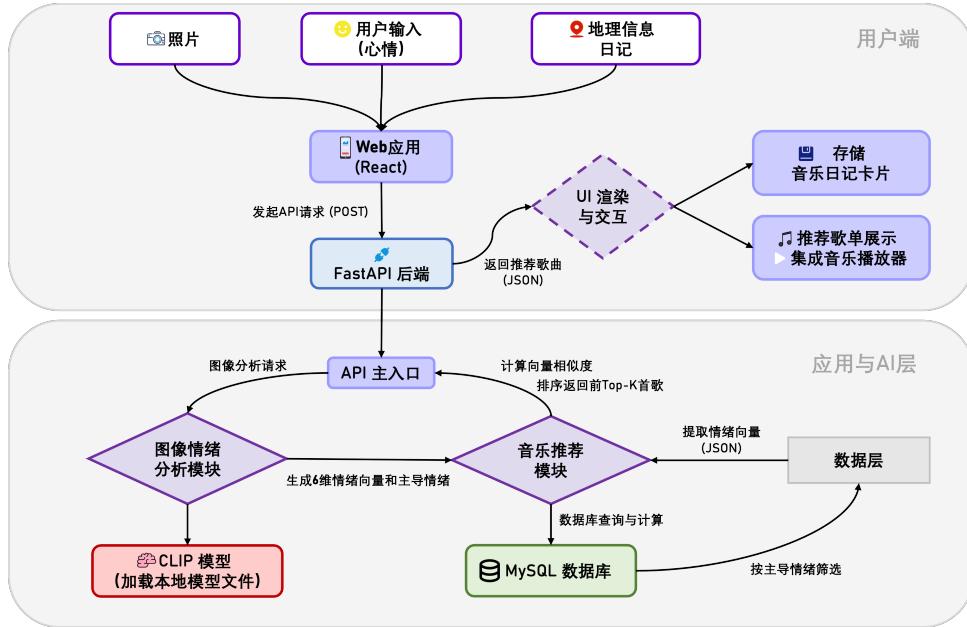


Figure 2: 架构总览

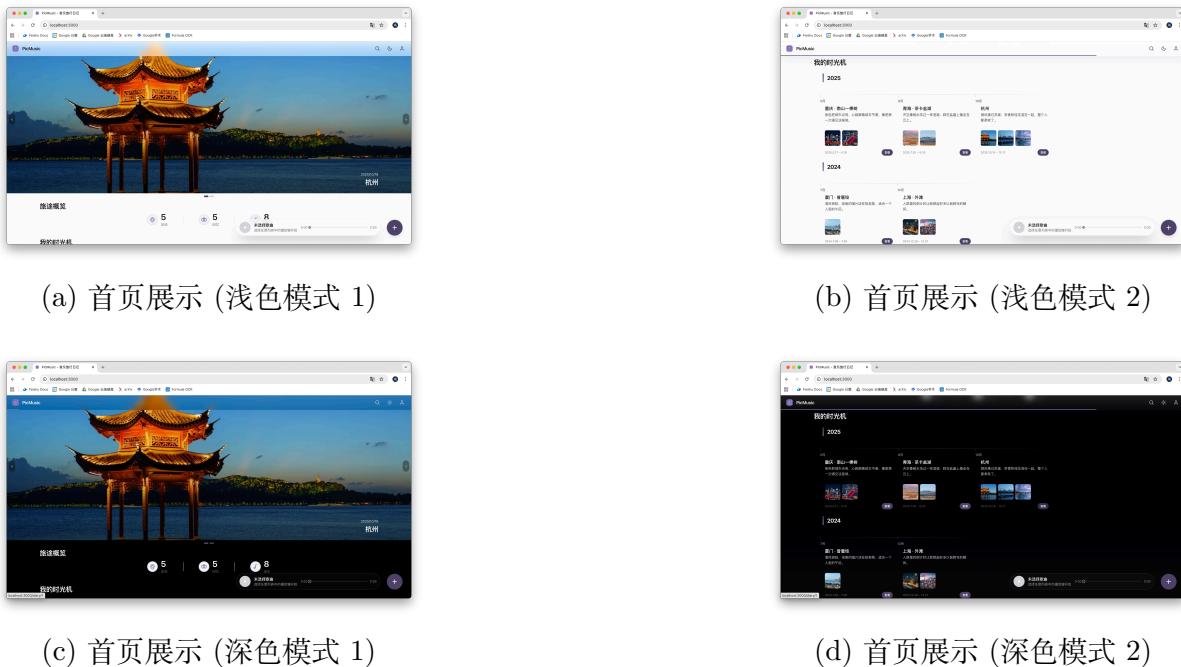


Figure 3: PicMusic 首页展示

### 3.1 一键获取歌曲推荐

这是本应用的功能，采用“图像情绪理解 + 歌曲情绪检索”的模式。

- 前端触发：**用户在新建日记页面（frontend/app/new/page.tsx）上传图片和填写基本信息后，点击“生成歌曲推荐”。系统将图片编码为 Data URL 存入 localStorage。
- API 调用：**客户端（frontend/app/new/loading/page.tsx）从 localStorage 读取 Data URL，并作为 image\_url 参数 POST 给 FastAPI 后端的 /recommend 接口。
- 结果处理：**后端返回 Top-K 歌曲列表（包含 id, name, artist），前端在推荐页面（frontend/app/new/recommend/page.tsx）展示结果，供用户试听和选择关联。

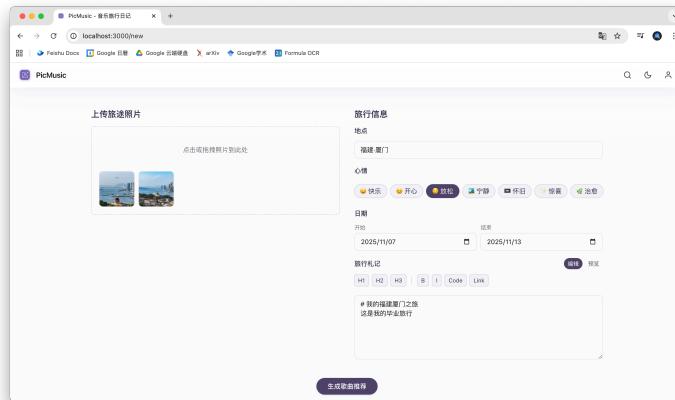
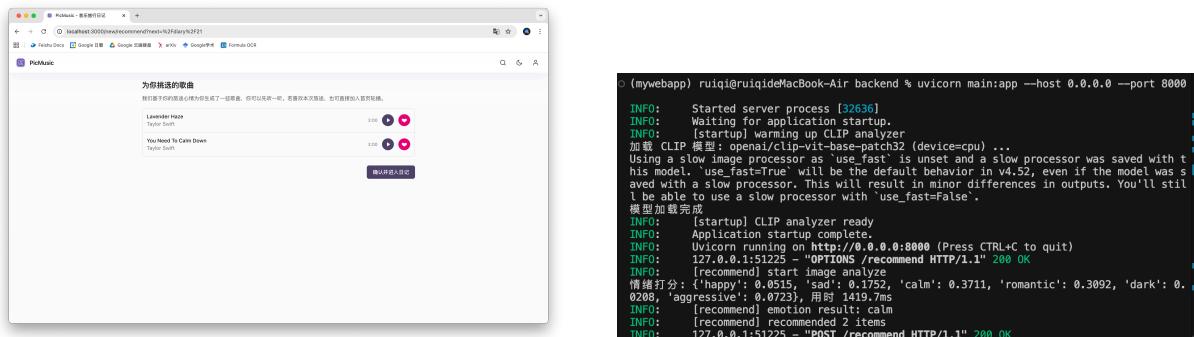


Figure 4: 用户上传图片并输入旅行信息



(a) 获得歌曲推荐（前台展示）

(b) 获得歌曲推荐（后台展示）

Figure 5: 歌曲推荐结果

## 3.2 旅途回忆可视化与编辑

- **可视化：**首页通过 `Timeline.tsx` 组件将日记按年份进行分组和可视化展示。
- **编辑与管理：**日记详情页（`frontend/components/DiaryEditor.tsx`）提供了一个集中编辑界面，支持修改所有字段（地点、心情、图片、文本、音乐列表）。特别地，用户支持 markdown 模式编辑文本，可读性更强。
- **收藏：**用户可以通过收藏按钮（`FavoriteToggle.tsx`）一键将日记加入首页轮播图展示，该轮播图由 `FeaturedCarouselClient.tsx` 筛选 `isFavorite` 字段为 ‘true’ 的日记。

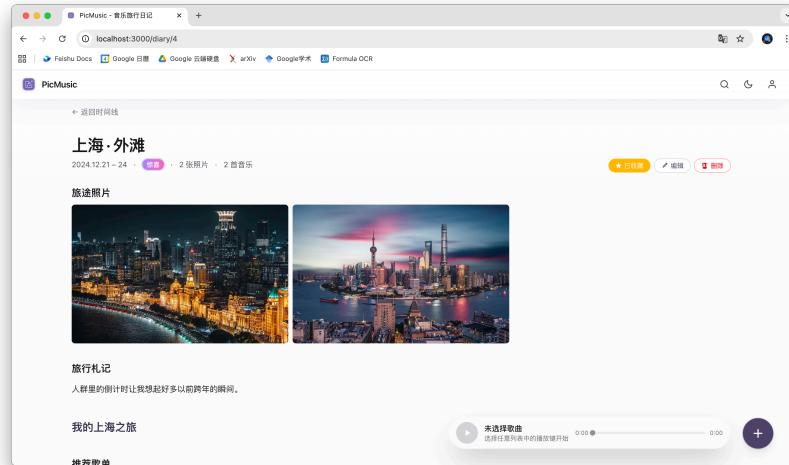


Figure 6: 旅行回忆展示

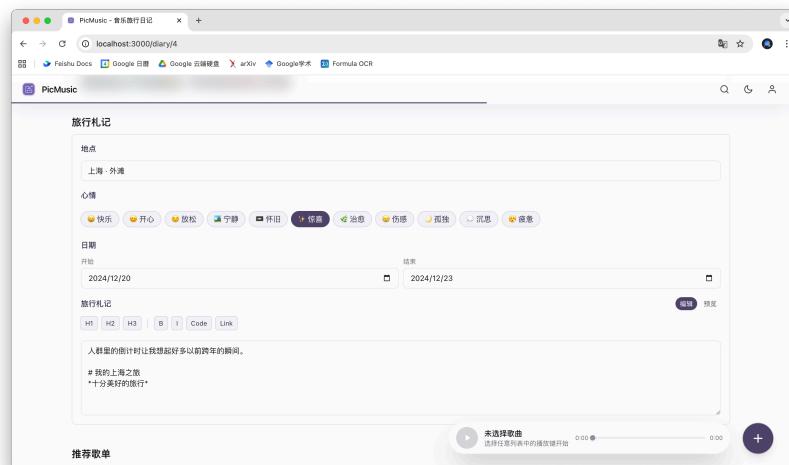


Figure 7: 旅行回忆编辑（支持 Markdown 编辑）

### 3.3 旅途回忆检索

应用提供快速搜索功能，允许用户根据**地点关键词**进行实时检索。前端 Header 组件通过防抖调用 Next.js API Route /api/diaries/search，后端查询数据库中 location 字段匹配的结果。

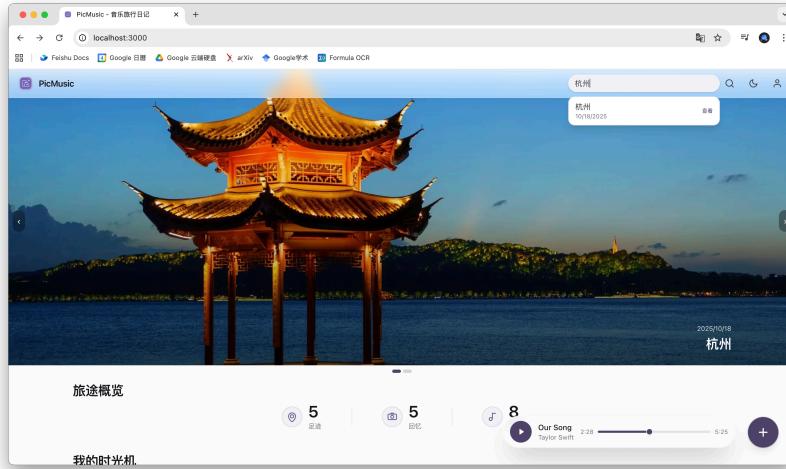


Figure 8: 搜索功能

### 3.4 个性化偏好定制

个人中心页面 (ProfileClient.tsx) 允许用户配置 MBTI 性格类型和音乐风格偏好。这些数据被存储在客户端 (localStorage)，用于未来将用户画像引入推荐算法的迭代工作。

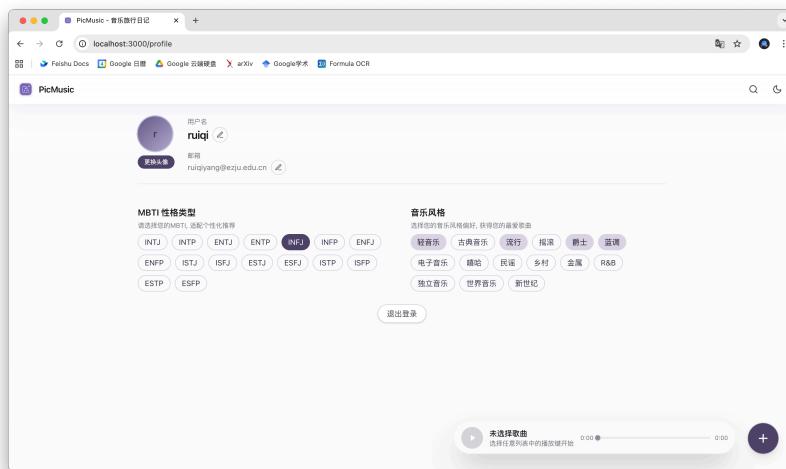


Figure 9: 个性化定制

## 4 主要技术详述与核心代码实现

### 4.1 前端 React + Next.js 技术

本项目选择 Next.js (TypeScript/React) 作为前端技术栈。这不仅利用了 React 的组件化优势，还通过 Next.js 的 App Router 提供了强大的路由管理和服务器组件能力，使数据获取（如 `getDiaries`）可以在服务器端高效完成。

### 4.2 智能体技术与推荐算法设计

#### 4.2.1 智能体技术：CLIP 图像情绪分析

后端的智能体由 Python 和 `transformers` 库实现，具体在 `backend/image_analyze.py` 中。该文件中的 `ImageEmotionAnalyzer` 类封装了 CLIP 模型。

`analyze` 方法是核心。它接收图片输入（支持 URL、Base64 Data URL 或本地路径），然后使用 `processor` 同时处理图片和预设的六个情绪文本标签 `MUSIC_EMOTION_LABELS`。模型 `outputs.logits_per_image` 输出了图像与每个标签的相似度分数，通过 `softmax` 归一化后，即得到代表该图像的情绪分布向量  $v_{img}$ 。

```
1 import io
2 import requests
3 from PIL import Image
4 import torch
5 from transformers import CLIPProcessor, CLIPModel
6 # ...
7
8 # 情绪标签（与曲库一致）
9 MUSIC_EMOTION_LABELS = [
10     "happy", "sad", "calm", "romantic", "dark", "aggressive"
11 ]
12
13 class ImageEmotionAnalyzer:
14     # ... (omitted __init__ model loading)
15
16     def analyze(self, image_input):
17         # ... (omitted image loading from URL/Base64)
18
19         # 计算图像对情绪标签的匹配分数
20         inputs = self.processor(
21             text=MUSIC_EMOTION_LABELS,
22             images=image,
23             return_tensors="pt",
24             padding=True
25         )
```

```

26
27     # ... (omitted device transfer)
28
29     with torch.no_grad():
30         outputs = self.model(**inputs)
31         # 使用 softmax 归一化 logits_per_image 得到概率分布
32         probs = outputs.logits_per_image.softmax(dim=1)[0].detach().cpu().numpy()
33
34         emotion_scores = {label: round(float(score), 4) for label, score in zip(MUSIC_
35             EMOTION_LABELS, probs)}
36         dominant_emotion = max(emotion_scores.items(), key=lambda x: x[1])[0]
37         # ...
38         return {
39             "emotion_scores": emotion_scores,
40             "dominant_emotion": dominant_emotion
41         }
42     # ...

```

Listing 1: 智能体技术核心代码实现

#### 4.2.2 推荐算法：余弦相似度

推荐算法在 `backend/recommend_songs.py` 中实现。首先，我们将歌曲在情绪空间中映射的多维向量  $v_{music}$ （存储于数据库 `emotion_json` 字段），与系统将用户输入图像经过 CLIP 提取的情绪标签向量  $v_{img}$ ，进行余弦相似度计算（见公式 1）。

余弦相似度公式：

$$\text{similarity}(v_{img}, v_{music}) = \frac{v_{img} \cdot v_{music}}{|v_{img}| |v_{music}|} \quad (1)$$

由于余弦相似度兼具计算效率与语义保留能力，适用于高维空间的快速匹配，因此能充分度量情绪语义的贴合程度。核心代码分析：`recommend_songs_by_emotion` 函数首先从数据库（`tracks` 表）中筛选出与图片 `dominant_emotion` 一致的歌曲，这是一个关键的性能优化步骤，避免了全库比较。然后，它遍历这些筛选后的歌曲，加载它们预先计算好的 `emotion_json` 向量 ( $v_{music}$ )，并高效计算  $v_{img}$  与每个  $v_{music}$  的相似度得分，最后排序返回 Top-K 结果。

```

1 import json
2 import numpy as np
3 from threadpoolctl import threadpool_limits
4 import mysql.connector
5 from sklearn.metrics.pairwise import cosine_similarity
6 # ...
7

```

```
8 def recommend_songs_by_emotion(emotion_json: dict, db_config: Dict[str, Any], top_k: int) -> list:
9     # 1. 转换输入向量 v_img (Image Vector)
10    input_vector = np.array([
11        emotion_json["emotion_scores"]["happy"],
12        # ... (omitted others)
13        emotion_json["emotion_scores"]["aggressive"]
14    ]).reshape(1, -1)
15
16    dominant_emotion = emotion_json["dominant_emotion"]
17    # ... (omitted db connection)
18
19    # 2. 数据库优化：仅查询主导情绪匹配的歌曲
20    query = (
21        "SELECT id, name, artist, emotion_json "
22        "FROM tracks "
23        "WHERE dominant_emotion = %s "
24        "AND emotion_json IS NOT NULL"
25    )
26    cursor.execute(query, (dominant_emotion,))
27    results = cursor.fetchall()
28    # ...
29
30    # 3. 计算余弦相似度
31    scored_songs = []
32    with threadpool_limits(limits=1): # 规避 BLAS 线程冲突
33        for row in results:
34            try:
35                # 转换向量 v_music (Song Vector)
36                song_vector_dict = json.loads(row["emotion_json"])
37                song_vector = np.array([
38                    song_vector_dict.get("happy", 0),
39                    # ... (omitted others)
40                ]).reshape(1, -1)
41
42                # 计算 v_img 和 v_music 的相似度 (Eq(1))
43                sim = cosine_similarity(input_vector, song_vector)[0][0]
44                scored_songs.append({
45                    # ... (omitted fields)
46                    "similarity": sim
47                })
48            except Exception:
49                continue
50
51    # 4. 排序和返回 Top-K
52    scored_songs.sort(key=lambda x: x["similarity"], reverse=True)
```

```
53     top_songs = [{"id": s["id"], "name": s["name"], "artist": s["artist"]} for s in
54     scored_songs[:top_k]]
55
55     return top_songs
```

Listing 2: 情感推荐算法核心代码实现

### 4.3 后端 FastAPI 技术

FastAPI 提供了轻量级、高性能的 Python API 服务。它使用 pydantic 模型 (RecommendRequest) 自动验证请求体，确保了 image\_url 和 top\_k 字段的类型正确。

@app.post("/recommend") 装饰器将 recommend\_songs 函数注册为 API 路由。该函数是核心的协调器：它接收 Pydantic 模型 req，首先调用 analyze\_image\_emotion 处理 req.image\_url，然后将情绪结果传递给 recommend\_songs\_by\_emotion 来查询数据库。`_RECOMMEND_LOCK` 确保了对 AI 模型的访问是线程安全的，防止并发请求导致资源冲突。

```
1 # ... (omitted imports and app setup)
2 _RECOMMEND_LOCK = threading.Lock()
3 # ...
4
5 # Pydantic 请求模型 (数据验证)
6 class RecommendRequest(BaseModel):
7     image_url: str
8     top_k: int = 3
9
10 # Pydantic 响应模型 (数据格式化)
11 class SongInfo(BaseModel):
12     id: int
13     name: str
14     artist: str
15
16 # 唯一公开接口: 推荐歌曲
17 @app.post("/recommend", response_model=List[SongInfo])
18 def recommend_songs(req: RecommendRequest):
19     try:
20         # 使用线程锁确保 AI 模型调用安全
21         with _RECOMMEND_LOCK:
22             logger.info("[recommend] start image analyze")
23             # 1. 调用 AI 智能体
24             emotion_result = analyze_image_emotion(req.image_url)
25             logger.info("[recommend] emotion result: %s", emotion_result.get("dominant
26             _emotion"))
```

```

27     # 2. 调用推荐算法
28     recommended = recommend_songs_by_emotion(emotion_result, db_config, top_k=
29         req.top_k)
30     logger.info("[recommend] recommended %d items", len(recommended))
31     return recommended
32
33 except Exception as e:
34     logger.exception("/recommend failed: %s", e)
35     # 失败时返回空列表，保证前端健壮性
36     return []

```

Listing 3: FastAPI 核心代码实现

## 4.4 数据库存储技术

项目使用 MySQL 存储大部分数据，Alibaba Cloud 存储图像等数据。前端通过 Next.js API Routes 作为后端 (BFF)，调用 frontend/lib/diaryRepo.ts 中的数据访问函数。diaryRepo.ts 使用 mysql2/promise 库与 MySQL 数据库通信，实现了对 diaries 表的增、查、改、删操作。

	<code>id</code>	<code>user_id</code>	<code>diary_datetime</code>	<code>trip_start</code>	<code>trip_end</code>	<code>location</code>	<code>mood</code>	<code>content</code>	<code>photo_urls_json</code>
1	1		2025-10-18 14:05:00	2025-10-14 00:00:00	2025-10-17 00:00:00	杭州 - 茶卡盐湖	宁静	微风拂过苏堤，茶香和桂花混在一起。[https://video-lyx.oss-cn-hangzhou.aliyuncs.com/m]	
2	1		2025-08-03 08:12:00	2025-07-30 10:00:00	2025-08-03 20:00:00	青海 - 茶卡盐湖	快乐	天空像被水洗过一样清澈，踩在盐晶上。	[https://video-lyx.oss-cn-hangzhou.aliyuncs.com/m]
3	1		2025-05-20 10:30:00	2025-05-17 00:00:00	2025-05-19 00:00:00	重庆 - 南山一棵树	激动	夜色把城市点亮，心跳跟着节奏跳。	[https://video-lyx.oss-cn-hangzhou.aliyuncs.com/m]
4	1		2024-1-31 23:50:00	2024-12-28 12:00:00	2024-12-31 23:59:00	上海 - 外滩	怀旧	人群里的倒计时让我想起好多以前跨年。	[https://video-lyx.oss-cn-hangzhou.aliyuncs.com/m]
5	1		2024-07-08 09:20:00	2024-07-06 08:00:00	2024-07-08 16:00:00	厦门 - 曾厝垵	孤独	海风很轻，街角的唱片店在放老歌。	[https://video-lyx.oss-cn-hangzhou.aliyuncs.com/m]
	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>

Figure 10: 数据库 diaries 表结构

	<code>id</code>	<code>name</code>	<code>artist</code>	<code>style</code>	<code>language</code>	<code>bpm</code>	<code>instrument</code>	<code>lyrics</code>	<code>label</code>	<code>emotion_json</code>
19293009		Our Song	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift 作曲：Taylor Swift I was ridin...	<code>NULL</code>	{"sad": 0.187, "calm": 0.125, "dark": 0.05}
50230386		Look What You Made Me Do	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 编曲：Jack Antonoff	<code>NULL</code>	{"sad": 0.24, "calm": 0.083, "dark": 0.05}
502307089		...Ready For It?	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：All Payne 作曲：Jack Antonoff/Schreier/Mart...	<code>NULL</code>	{"sad": 0.057, "calm": 0.125, "dark": 0.05}
516191336		...I'm Ready	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Martin Shelliack 作曲：T...	<code>NULL</code>	{"sad": 0.118, "calm": 0.035, "dark": 0.05}
516823205		Don't Blame Me	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Max Martin/Shelliack 作曲：T...	<code>NULL</code>	{"sad": 0.182, "calm": 0.091, "dark": 0.05}
1382778514		Lover	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift 作曲：Taylor Swift 编曲：Taylor ...	<code>NULL</code>	{"sad": 0.062, "calm": 0.187, "dark": 0.05}
138277829		The Archer	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Luck Antonoff 作曲：Taylor ...	<code>NULL</code>	{"sad": 0.094, "calm": 0.087, "dark": 0.05}
1382778973		You Need To Calm Down	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Joel Little/Taylor Swift 作曲：Joel Little/T...	<code>NULL</code>	{"sad": 0.133, "calm": 0.267, "dark": 0.05}
138278100		The Man	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Joel Little/Taylor Swift 作曲：Joel Little/T...	<code>NULL</code>	{"sad": 0.316, "calm": 0.105, "dark": 0.05}
138278139		Miss Americana & The Heartbreak Prince	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 作曲：Taylor S...	<code>NULL</code>	{"sad": 0.222, "calm": 0.111, "dark": 0.05}
1382781434		Death By A Thousand Cuts	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 作曲：Taylor S...	<code>NULL</code>	{"sad": 0.333, "calm": 0.125, "dark": 0.05}
1465111714		the 1	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 作曲：Taylor S...	<code>NULL</code>	{"sad": 0.273, "calm": 0.182, "dark": 0.05}
1465111917		august	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 作曲：Taylor S...	<code>NULL</code>	{"sad": 0.286, "calm": 0.19, "dark": 0.05}
1465111971		betty	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	编曲：Taylor Swift/William Bowery 作词：Taylor ...	<code>NULL</code>	{"sad": 0.261, "calm": 0.174, "dark": 0.05}
1465114419		cardigan	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Aaron Dessner 作曲：Taylor ...	<code>NULL</code>	{"sad": 0.222, "calm": 0.144, "dark": 0.05}
1465114457		the last great american dynasty	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Aaron Dessner 作曲：Taylor ...	<code>NULL</code>	{"sad": 0.235, "calm": 0.118, "dark": 0.05}
1465114469		my tears rochoet	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift 作曲：Taylor Swift We gather...	<code>NULL</code>	{"sad": 0.32, "calm": 0.08, "dark": 0.05}
1465114515		illicit affairs	Taylor Swift	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	<code>NULL</code>	作词：Taylor Swift/Jack Antonoff 作曲：Taylor S...	<code>NULL</code>	{"sad": 0.269, "calm": 0.115, "dark": 0.05}

Figure 11: 数据库 tracks 表结构 (含情绪向量)

### 4.4.1 Create (增) - createDiary

- API 路由: POST /api/diaries
- Repo 函数: createDiary in diaryRepo.ts

- 代码分析：

- (1). frontend/app/api/diaries/route.ts 中的 POST 函数负责接收前端新建日记的 fetch 请求。
- (2). 它解析 req.json() 得到日记内容 (location, mood, text, photos 等)。
- (3). 调用 createDiary 函数，将 JS 对象转换为数据库格式 (例如，将 photos 数组 JSON.stringify 为字符串，将 date 转换为 SQL DATETIME 格式)。
- (4). createDiary 函数执行 INSERT INTO diaries ... SQL 语句，将新日记存入数据库。

```
1 import { NextRequest, NextResponse } from "next/server";
2 import { createDiary, getDiaries, getDiaryById } from "@/lib/diaryRepo";
3
4 // ...
5 // POST /api/diaries
6 export async function POST(req: NextRequest) {
7   const body = await req.json().catch(() => ({}));
8   const {
9     date,
10    startDate,
11    endDate,
12    location = "",
13    mood = "快乐",
14    text = "",
15    photos = [],
16    trackIds = [],
17    isFavorite = false,
18 } = body || {};
19
20 if (!date) return NextResponse.json({ error: "date required" }, { status: 400 });
21
22 // 1. 调用 Repo 函数
23 const res = await createDiary(1, {
24   diary_datetime: new Date(date).toISOString().slice(0, 19).replace("T", " "),
25   trip_start: startDate ? new Date(startDate).toISOString().slice(0, 19).replace("T",
26     " ") : null,
27   trip_end: endDate ? new Date(endDate).toISOString().slice(0, 19).replace("T", " ")
28     : null,
29   location,
30   mood,
31   content: text,
32   photo_urls_json: JSON.stringify(photos || []),
33   track_ids_json: JSON.stringify(trackIds || []),
34   is_favorite: !!isFavorite,
```

```
33 });
34 // ... (omitted response handling)
35 }
```

Listing 4: picmusic\_web/frontend/app/api/diaries/route.ts (POST)

```
1 import { getPool } from "@lib/db";
2 import type { Diary } from "@lib/types";
3 import { getTracksMapByIds } from "@lib/tracksRepo";
4 // ... (omitted helpers)
5
6 export async function createDiary(userId: number, data: {
7   diary_datetime: string;
8   trip_start?: string | null;
9   trip_end?: string | null;
10  location: string;
11  mood: string;
12  content: string;
13  photo_urls_json?: string; // JSON
14  track_ids_json?: string; // JSON
15  is_favorite?: boolean;
16 }) {
17   const pool = getPool();
18   // 2. 执行 INSERT SQL 语句
19   const [res] = await pool.query(
20     `INSERT INTO diaries (user_id, diary_datetime, trip_start, trip_end, location,
21       mood, content, photo_urls_json, track_ids_json, is_favorite)
22       VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
23     [
24       userId,
25       data.diary_datetime,
26       data.trip_start ?? null,
27       data.trip_end ?? null,
28       data.location,
29       data.mood,
30       data.content,
31       data.photo_urls_json ?? "[]",
32       data.track_ids_json ?? "[]",
33       data.is_favorite ? 1 : 0,
34     ]
35   );
36   return res as any;
37 }
```

Listing 5: picmusic\_web/frontend/lib/diaryRepo.ts (createDiary)

#### 4.4.2 Read (查) - getDiaryById

- API 路由: GET /api/diaries/[id]
- Repo 函数: getDiaryById in diaryRepo.ts
- 代码分析:
  - (1). frontend/app/api/diaries/[id]/route.ts 中的 GET 函数通过 ctx.params 获取 URL 中的动态 ID。
  - (2). 调用 getDiaryById, 该函数执行 SELECT ... FROM diaries WHERE id = ? ... 查询。
  - (3). getDiaryById 查询到日记后, 解析 track\_ids\_json, 并调用 getTracksMapByIds (来自 tracksRepo.ts) 发起第二次数据库查询, 以获取关联歌曲的详细信息 (如歌名、艺术家)。
  - (4). 这种 “N+1 查询” 的优化 (在此例中是 1+1, 批量获取) 确保了返回给前端的日记对象包含了完整的 tracks 数组。

```
1 import { NextRequest, NextResponse } from "next/server";
2 import { deleteDiary, getDiaryById, updateDiary } from "@/lib/diaryRepo";
3 import { CURRENT_USER_ID } from "@/lib/config";
4
5 // GET /api/diaries/[id]
6 export async function GET(_: NextRequest, ctx: { params: Promise<{ id: string }> }) {
7   const { id: idStr } = await ctx.params;
8   const id = Number.parseInt(idStr, 10);
9   if (!Number.isFinite(id)) return NextResponse.json({ error: "invalid id" }, { status
10    : 400 });
11
12  // 1. 调用 Repo 函数
13  const row = await getDiaryById(id, CURRENT_USER_ID);
14
15  if (!row) return NextResponse.json({ error: "not found" }, { status: 404 });
16  return NextResponse.json(row);
17 }
```

Listing 6: picmusic\_web/frontend/app/api/diaries/id/route.ts (GET)

```
1 // ... (omitted imports and helpers)
2 export async function getDiaryById(id: number, userId = 1): Promise<(Diary & {
3   isFavorite?: boolean } | null> {
4   const pool = getPool();
5   // 2. 执行 SELECT SQL 语句
```

```
5  const [rows] = await pool.query(
6    `SELECT id, user_id, diary_datetime, trip_start, trip_end, location, mood, content
7      ,
8        photo_urls_json, track_ids_json, is_favorite
9      FROM diaries
10     WHERE id = ? AND user_id = ?
11     LIMIT 1`,
12   [id, userId]
13 );
14 const r = (rows as any[])[0];
15 if (!r) return null;
16 const d = rowToDiary(r); // 转换基本字段
17 const ids = parseIdArrayAsStrings(r.track_ids_json);
18
19 if (ids.length) {
20   // 3. 关联查询：获取歌曲详情
21   const trackMap = await getTracksMapByIds(ids);
22   const tracks = ids.map((tid) => trackMap.get(String(tid))).filter(Boolean) as any[]
23   ;
24   (d as any).tracks = tracks;
25 }
```

Listing 7: picmusic\_web/frontend/lib/diaryRepo.ts (getDiaryById)

#### 4.4.3 Update (改) - updateDiary

- API 路由: PUT /api/diaries/[id]
- Repo 函数: updateDiary in diaryRepo.ts
- 代码分析:
  - (1). frontend/app/api/diaries/[id]/route.ts 中的 PUT 函数接收 DiaryEditor.tsx 的“保存修改”请求。
  - (2). 它动态地检查请求 body 中包含哪些字段 (如 location, mood, photos 等), 并构建一个 patch 对象。
  - (3). updateDiary 函数接收这个 patch 对象, 遍历其键值对, 动态构建 UPDATE diaries SET field1 = ?, field2 = ? ... SQL 语句。

```
1 // ...
2 // PUT /api/diaries/[id]
```

```
3 export async function PUT(req: NextRequest, ctx: { params: Promise<{ id: string }> })
4 {
5     const { id: idStr } = await ctx.params;
6     const id = Number.parseInt(idStr, 10);
7     if (!Number.isFinite(id)) return NextResponse.json({ error: "invalid id" }, { status
8         : 400 });
9
10    // 1. 动态构建 patch 对象
11    if (body.date) patch.diary_datetime = new Date(body.date).toISOString().slice(0, 19)
12        .replace("T", " ");
13    if ("startDate" in body) patch.trip_start = body.startDate ? new Date(body.startDate
14        ).toISOString().slice(0, 19).replace("T", " ") : null;
15    if ("endDate" in body) patch.trip_end = body.endDate ? new Date(body.endDate).
16        toISOString().slice(0, 19).replace("T", " ") : null;
17    if ("location" in body) patch.location = body.location ?? "";
18    if ("mood" in body) patch.mood = body.mood ?? "快乐";
19    if ("text" in body) patch.content = body.text ?? "";
20    if ("photos" in body) patch.photo_urls_json = JSON.stringify(body.photos || []);
21    if ("trackIds" in body) patch.track_ids_json = JSON.stringify(body.trackIds || []);
22    if ("isFavorite" in body) patch.is_favorite = body.isFavorite ? 1 : 0;
23
24    // 2. 调用 Repo 函数
25    const res: any = await updateDiary(id, CURRENT_USER_ID, patch);
26    // ... (omitted response handling)
27}
```

Listing 8: picmusic\_web/frontend/app/api/diaries/id/route.ts (PUT)

```
1 // ...
2 export async function updateDiary(
3     id: number,
4     userId: number,
5     payload: Partial<{
6         // ... (omitted types)
7     }>
8 ) {
9     const pool = getPool();
10    const fields: string[] = [];
11    const values: any[] = [];
12
13    // 3. 动态构建 SET 子句
14    for (const [k, v] of Object.entries(payload)) {
15        fields.push(`[${k}] = ?`);
16        values.push(v);
17    }
```

```
18 if (!fields.length) return { affectedRows: 0 } as any;
19
20 values.push(id, userId);
21 // 4. 执行 UPDATE SQL 语句
22 const [res] = await pool.query(
23   `UPDATE diaries SET ${fields.join(", ")} WHERE id = ? AND user_id = ?`,
24   values
25 );
26 return res as any;
27 }
```

Listing 9: picmusic\_web/frontend/lib/diaryRepo.ts (updateDiary)

#### 4.4.4 Delete (删) - deleteDiary

- API 路由: DELETE /api/diaries/[id]
- Repo 函数: deleteDiary in diaryRepo.ts
- 代码分析:
  - (1). frontend/app/api/diaries/[id]/route.ts 中的 DELETE 函数是此操作的入口。
  - (2). 它从 URL 中解析 id，并调用 deleteDiary 函数。
  - (3). deleteDiary 函数执行 DELETE FROM diaries WHERE id = ? AND user\\_id = ? SQL 语句。
  - (4). Where 子句中的 user\\_id = ? 是关键的安全约束，确保用户只能删除自己的数据。

```
1 // ...
2 // DELETE /api/diaries/[id]
3 export async function DELETE(_: NextRequest, ctx: { params: Promise<{ id: string }> }) {
4   const { id: idStr } = await ctx.params;
5   const id = Number.parseInt(idStr, 10);
6   if (!Number.isFinite(id)) return NextResponse.json({ error: "invalid id" }, { status: 400 });
7
8   // 1. 调用 Repo 函数执行删除
9   await deleteDiary(id, CURRENT_USER_ID);
10  return NextResponse.json({ ok: true });
11 }
```

Listing 10: picmusic\_web/frontend/app/api/diaries/id/route.ts (DELETE)

```
1 // ...
2 export async function deleteDiary(id: number, userId = 1) {
3   const pool = getPool();
4   // 2. 执行 SQL 删除
5   const [res] = await pool.query(`DELETE FROM diaries WHERE id = ? AND user_id = ?`, [
6     id, userId]);
7   return res as any;
}
```

Listing 11: picmusic\_web/frontend/lib/diaryRepo.ts (deleteDiary)

## 5 总结与展望

PicMusic 项目成功地实现了核心的多模态推荐功能，通过结合 **FastAPI 的高性能 API 架构、CLIP 模型的情绪识别能力和余弦相似度算法**，解决了“图片检索音乐”的挑战，并顺利完成了实验目标。前端采用的 Next.js 和事件总线设计（GlobalAudioBar）确保了流畅的单页应用体验，而基于 MySQL 的数据层则稳固地支持了日记数据的 CRUD 和检索需求。

未来可以在以下方面进行改进和扩展：

- (1). **多模态深度融合：**当前推荐主要依赖图像情绪。下一步可将日记文本内容（content）的情绪分析结果（例如利用 lyrics\_analyze.py 中类似的情感分析技术）与图像情绪向量进行加权融合，形成更全面的“旅行记忆情绪”向量。
- (2). **用户画像个性化：**将用户在个人中心设置的偏好（MBTI、音乐风格）引入推荐模型，通过调整余弦相似度的权重或使用混合模型，进一步提升推荐的个性化程度。