巅峰态 (PeakState) 技术实施文档

文档版本： 2.0
创建日期： 2025年10月8日
作者： Manus AI
适用对象： 技术团队、AI编程工具（Claude Code、Cursor、Codex）

目录

---

1. 系统架构设计

1.1 整体架构图

```mermaid graph TB subgraph "客户端层 - Client Layer" A1[iOS App - React Native] A2[Android App - React Native] A3[Web App - 可选] end
```

```
 subgraph "API网关层 - API Gateway"
    B1[Nginx / ALB]
    B2[API Gateway]
    B3[认证中间件]
end

subgraph "应用服务层 - Application Layer"
    C1[用户服务<br/>User Service]
    C2[对话服务<br/>Chat Service]
    C3[数据采集服务<br/>Data Collection]
    C4[分析服务<br/>Analytics Service]
    C5[推送服务<br/>Notification Service]
end

subgraph "AI智能层 - AI Layer"
    D1[LLM服务<br/>GPT-4 API]
    D2[精力预测模型<br/>Energy Prediction]
    D3[推荐引擎<br/>Recommendation]
    D4[情感分析<br/>Sentiment Analysis]
end

subgraph "数据层 - Data Layer"
    E1[(PostgreSQL<br/>用户数据)]
    E2[(MongoDB<br/>对话历史)]
    E3[(Redis<br/>缓存)]
    E4[(时序数据库<br/>InfluxDB)]
end

subgraph "外部服务 - External Services"
    F1[Apple Health API]
    F2[Google Fit API]
    F3[日历API]
    F4[天气API]
    F5[支付网关]
end

A1 --> B1
A2 --> B1
A3 --> B1
B1 --> B2
B2 --> B3
B3 --> C1
B3 --> C2
B3 --> C3
B3 --> C4
B3 --> C5

C2 --> D1
C4 --> D2
C4 --> D3
C2 --> D4

C1 --> E1
C2 --> E2
C3 --> E4
C4 --> E1
C1 --> E3
C2 --> E3

C3 --> F1
C3 --> F2
C3 --> F3
C3 --> F4
C1 --> F5
```

```
```

1.2 技术栈详细说明

层级	组件	技术选型	理由
前端	移动应用	React Native	跨平台开发，一套代码覆盖 iOS/Android
	状态管理	Redux + Redux Toolkit	统一状态管理，便于调试
	网络请求	Axios	支持拦截器，易于处理认证
	本地存储	AsyncStorage + react-native-keychain	安全存储敏感数据
后端	API框架	FastAPI (Python)	高性能，原生支持异步，易于集成AI
	认证	JWT + OAuth 2.0	无状态认证，支持第三方登录
	任务队列	Celery + Redis	处理异步任务（数据同步、推送等）
AI	LLM	OpenAI GPT-4 API	最强对话能力
	ML框架	scikit-learn + XGBoost	传统ML模型
	DL框架	PyTorch	深度学习模型（后期）
数据库	关系型	PostgreSQL	可靠性高，支持JSON字段
	文档型	MongoDB	灵活的对话历史存储
	缓存	Redis	高性能缓存和消息队列
	时序	InfluxDB	专为时序数据优化
部署	云平台	阿里云	中国大陆用户，合规性好
	容器化	Docker + Docker Compose	环境一致性
	编排	Kubernetes (可选)	后期扩展
	CI/CD	GitHub Actions	自动化部署

1.3 数据库设计

1.3.1 PostgreSQL 核心表结构

```sql -- 用户表 CREATE TABLE users ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), email VARCHAR(255) UNIQUE NOT NULL, phone VARCHAR(20), password_hash VARCHAR(255) NOT NULL, name VARCHAR(100), avatar_url VARCHAR(500), coach_type VARCHAR(50) DEFAULT 'balanced', -- 教练风格 subscription_status VARCHAR(20) DEFAULT 'trial', -- trial, active, expired subscription_end_date TIMESTAMP, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP );

-- 用户画像表 CREATE TABLE user_profiles ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, age INTEGER, gender VARCHAR(10), occupation VARCHAR(100), goals JSONB, -- ["提升工作效率", "改善睡眠"] challenges JSONB, -- ["经常熬夜", "压力大"] sleep_preference JSONB, -- {"target_hours": 8, "bedtime": "23:00"} work_schedule JSONB, -- {"work_days": [1,2,3,4,5], "work_hours": "9-18"} created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP );

-- 精力记录表 (时序数据，也会同步到InfluxDB) CREATE TABLE energy_records ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, timestamp TIMESTAMP NOT NULL, energy_level INTEGER CHECK (energy_level BETWEEN 1 AND 10), energy_source VARCHAR(20), -- 'predicted', 'user_reported', 'calculated' context JSONB, -- {"activity": "meeting", "location": "office"} created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, INDEX idx_user_timestamp (user_id, timestamp DESC) );

-- 睡眠数据表 CREATE TABLE sleep_records ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, sleep_date DATE NOT NULL, bedtime TIMESTAMP, wake_time TIMESTAMP, duration_hours DECIMAL(4,2), quality_score INTEGER CHECK (quality_score BETWEEN 1 AND 10), deep_sleep_hours DECIMAL(4,2), rem_sleep_hours DECIMAL(4,2), data_source VARCHAR(50), -- 'apple_health', 'google_fit', 'manual' raw_data JSONB, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, UNIQUE(user_id, sleep_date) );

-- 活动数据表 CREATE TABLE activity_records ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, date
```

DATE NOT NULL, steps INTEGER, active_minutes INTEGER, calories_burned INTEGER, exercise_sessions JSONB, -- [{"type": "running", "duration": 30}] data_source VARCHAR(50), created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, UNIQUE(user_id, date));

-- 干预记录表 CREATE TABLE interventions (id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, timestamp TIMESTAMP NOT NULL, intervention_type VARCHAR(50), -- 'breathing', 'meditation', 'exercise' duration_seconds INTEGER, completion_status VARCHAR(20), -- 'completed', 'partial', 'skipped' pre_energy_level INTEGER, post_energy_level INTEGER, user_feedback JSONB, -- {"helpful": true, "rating": 4} created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

-- 推荐历史表 CREATE TABLE recommendation_history (id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES users(id) ON DELETE CASCADE, timestamp TIMESTAMP NOT NULL, recommendation_type VARCHAR(50), recommendation_content JSONB, user_action VARCHAR(20), -- 'accepted', 'dismissed', 'ignored' effectiveness_score DECIMAL(3,2), -- 0-1, 事后评估 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP); ```

1.3.2 MongoDB 对话历史结构

```javascript // conversations 集合 { "_id": ObjectId("..."), "user_id": "uuid", "conversation_id": "uuid", "started_at": ISODate("2025-10-08T08:00:00Z"), "last_message_at": ISODate("2025-10-08T08:15:00Z"), "conversation_type": "morning_briefing", // morning_briefing, evening_review, ad_hoc "messages": [{ "role": "assistant", "content": "早上好！我看到你昨晚睡了7.5小时，质量不错。今天预计精力状态良好。", "timestamp": ISODate("2025-10-08T08:00:00Z"), "metadata": { "energy_prediction": 8, "sleep_analysis": {...} } }, { "role": "user", "content": "但我还是觉得有点累", "timestamp": ISODate("2025-10-08T08:01:00Z") }, { "role": "assistant", "content": "我理解。虽然睡眠时长足够，但可能睡眠质量还有提升空间。要不要试试5分钟的呼吸练习？", "timestamp": ISODate("2025-10-08T08:01:30Z"), "metadata": { "intervention_suggested": "breathing", "reasoning": "用户主观感受与预测不符，建议快速干预" } }], "summary": "用户睡眠时长足够但主观感受疲劳，建议呼吸练习", "tags": ["sleep_quality", "fatigue", "breathing_exercise"] } ```

```

2. 数据流程设计

2.1 数据采集流程

```mermaid sequenceDiagram participant U as 用户设备 participant A as APP participant DC as 数据采集服务 participant EXT as 外部API participant DB as 数据库 participant TS as 时序数据库

```
 Note over U,TS: 被动数据采集（后台自动）

A->>DC: 定时任务触发（每小时）
DC->>EXT: 请求Apple Health数据
EXT-->>DC: 返回睡眠、心率、运动数据
DC->>DC: 数据清洗和验证
DC->>DB: 存储到PostgreSQL
DC->>TS: 存储到InfluxDB（时序）

Note over U,TS: 主动数据采集（用户触发）

U->>A: 用户与AI教练对话
A->>DC: 提交对话内容
DC->>DC: 提取主观数据（精力评分、情绪）
DC->>DB: 存储精力记录
DC->>TS: 更新时序数据
```

```

2.2 AI对话流程

```mermaid sequenceDiagram participant U as 用户 participant APP as 前端APP participant CS as 对话服务 participant AS as 分析服务 participant LLM as GPT-4 API participant DB as 数据库
```

```
 U->>APP: 发送消息
APP->>CS: POST /api/chat/message

CS->>DB: 获取用户画像
CS->>DB: 获取最近精力数据
CS->>DB: 获取对话历史

CS->>AS: 请求当前精力评估
AS->>AS: 运行预测模型
AS-->>CS: 返回精力评估结果

CS->>CS: 构建上下文Prompt
Note over CS: 包含：用户画像、精力数据、<br/>对话历史、当前评估

CS->>LLM: 发送Prompt
LLM-->>CS: 返回AI回复

CS->>DB: 保存对话记录
CS-->>APP: 返回AI回复
APP-->>U: 显示回复

opt 如果AI建议干预
    CS->>CS: 记录推荐
    CS->>APP: 推送干预建议
end
```

```

## 2.3 精力预测流程

```mermaid flowchart TD A[触发预测] --> B{用户数据充足?} B -->|否| C[使用通用模型] B -->|是| D[加载个性化模型]

```
 C --> E[收集特征数据]
D --> E

E --> F[特征工程]
F --> G[模型推理]

G --> H{预测精力水平}
H --> I[高精力 8-10]
H --> J[中精力 5-7]
H --> K[低精力 1-4]

I --> L[生成建议:
保持状态]
J --> M[生成建议:
适度干预]
K --> N[生成建议:
立即干预]

L --> O[返回预测结果]
M --> O
N --> O

O --> P[更新缓存]
P --> Q[记录预测历史]
```

```

---
```

3. 用户交互流程

3.1 用户旅程地图

```mermaid journey title 用户一天的精力管理旅程 section 早晨 (7:00-9:00) 收到晨间简报推送: 5: 用户 打开APP查看简报: 5: 用户 与AI教练对话: 4: 用户 授权数据同步: 3: 用户 section 上午 (9:00-12:00) 专注工作: 4: 用户 收到精力预警: 3: 用户 进行呼吸练习: 4: 用户 section 中午 (12:00-14:00) 午餐休息: 5: 用户 查看精力仪表盘: 3: 用户 section 下午 (14:00-18:00) 继续工作: 3: 用户 收到专注提醒: 4: 用户 使用专注计时器: 5: 用户 section 晚上 (18:00-22:00) 下班运动: 5: 用户 收到晚间复盘推送: 4: 用户 与AI教练复盘: 5: 用户 设定明日目标: 4: 用户 section 睡前 (22:00-23:00) 收到睡眠提醒: 5: 用户 进行冥想练习: 4: 用户 入睡: 5: 用户 ```

3.2 核心交互流程

3.2.1 首次使用流程

```mermaid flowchart TD A[下载APP] --> B[启动APP] B --> C[欢迎动画] C --> D[选择教练形象] D --> E[AI教练自我介绍]

```
 E --> F[请求数据授权]
F --> G{用户授权?}
G -->|是| H[连接Apple Health/Google Fit]
G -->|否| I[说明无硬件方案]

H --> J[初始问卷 1/5:
基本信息]
I --> J

J --> K[初始问卷 2/5:
目标设定]
K --> L[初始问卷 3/5:
当前挑战]
L --> M[初始问卷 4/5:
生活习惯]
M --> N[初始问卷 5/5:
工作模式]

N --> O[生成个性化画像]
O --> P[展示订阅选项]
P --> Q{选择订阅?}

Q -->|7天试用| R[绑定支付方式]
O -->|直接付费| R
Q -->|暂不订阅| S[进入免费模式
功能受限]

R --> T[完成入职]
S --> T
T --> U[进入主界面]
```

```
```

3.2.2 晨间简报流程

```mermaid sequenceDiagram participant S as 系统定时任务 participant AS as 分析服务 participant CS as 对话服务 participant NS as 推送服务 participant U as 用户

```
Note over S: 每天早上7:00触发

S->>AS: 生成晨间分析
AS->>AS: 分析昨晚睡眠
AS->>AS: 预测今日精力
AS->>AS: 检查今日日程
AS-->>S: 返回分析结果

S->>CS: 生成晨间简报内容
CS->>CS: 构建个性化简报
CS-->>S: 返回简报文本

S->>NS: 发送推送通知
NS->>U: 推送："早上好！今天的精力简报已准备好"

U->>APP: 点击推送打开APP
APP->>CS: 请求晨间简报
CS-->>APP: 返回完整简报
APP->>U: 展示简报内容

opt 用户与AI互动
    U->>APP: 发送问题
    APP->>CS: 继续对话
    CS-->>APP: AI回复
end
```

```

---

4. 核心功能模块详细设计

4.1 用户认证模块

功能描述： 处理用户注册、登录、JWT认证

技术方案： - JWT Token认证 - Refresh Token机制 - 第三方登录（Apple、微信）

API设计：

```python
```

POST /api/auth/register

{ "email": "user@example.com", "password": "SecurePass123!", "name": "张三" }

Response

{ "user_id": "uuid", "access_token": "jwt_token", "refresh_token": "refresh_token", "expires_in": 3600 }

POST /api/auth/login

{ "email": "user@example.com", "password": "SecurePass123!" }

POST /api/auth/refresh

{ "refresh_token": "refresh_token" } ```

4.2 数据采集模块

功能描述： 从多个数据源采集用户精力相关数据

数据源： 1. Apple Health / Google Fit 2. 系统日历 3. 用户对话（主观数据） 4. 第三方API（天气等）

采集策略： - 后台定时任务（每小时） - 用户触发（打开APP时） - 实时同步（对话中）

API设计：

```python
```

POST /api/data/sync/health

同步健康数据

{ "data_source": "apple_health", "data_type": "sleep", "records": [{ "date": "2025-10-07", "bedtime": "2025-10-07T23:00:00Z", "wake_time": "2025-10-08T07:00:00Z", "duration_hours": 8.0, "deep_sleep_hours": 2.5, "rem_sleep_hours": 1.5 }] }

POST /api/data/energy/report

用户手动报告精力

{ "energy_level": 7, "timestamp": "2025-10-08T10:00:00Z", "context": { "activity": "working", "mood": "focused" } } ```

4.3 AI对话模块

功能描述： 提供智能对话能力，模拟真人教练

核心组件： 1. 上下文管理器 2. Prompt构建器 3. LLM接口封装 4. 对话历史管理

Prompt模板：

```python COACH_SYSTEM_PROMPT = """ 你是{user_name}的私人精力管理教练，名字叫"小峰"。

## 你的角色定位

- 专业：你拥有精力管理、睡眠科学、运动营养等领域的专业知识
- 共情：你能理解用户的困难和挑战，给予情感支持
```

- 行动导向：你的建议都是具体的、可执行的
- 个性化：你了解用户的背景、目标和习惯

用户画像

- 姓名：{user_name}
- 年龄：{age}岁
- 职业：{occupation}
- 目标：{goals}
- 挑战：{challenges}

当前数据

- 昨晚睡眠：{sleep_last_night}小时（质量评分：{sleep_quality}/10）
- 当前精力预测：{current_energy}/10
- 今日精力趋势：{energy_trend}
- 今日重要事项：{today_schedule}

对话历史

{conversation_history}

你的回复要求

1. 语气：温和、专业、鼓励
2. 长度：简洁明了，一般2-3句话
3. 结构：先共情，再分析，后建议
4. 可操作性：给出具体的行动建议

用户刚刚说："{user_message}"

请回复： """ ```

API设计：

```python

# POST /api/chat/message

{ "message": "我今天感觉很累", "conversation_id": "uuid", # 可选，新对话则不传 "context": { "trigger": "user_initiated" # user_initiated, morning_briefing, intervention } }

# Response

{ "conversation_id": "uuid", "message_id": "uuid", "response": "我注意到你昨晚只睡了5.5小时，这可能是你感到疲劳的主要原因。下午有个重要会议，要不要现在做个5分钟的呼吸练习，帮你恢复一下精力？", "metadata": { "energy_assessment": 4, "intervention_suggested": { "type": "breathing", "duration": 300, "reason": "低精力状态，即将有重要任务" } }, "timestamp": "2025-10-08T10:05:00Z" }
```

4.4 精力预测模块

功能描述： 基于历史数据预测用户未来的精力状态

模型架构（MVP阶段）：

```python class EnergyPredictionModel: """ 精力预测模型（MVP版本：基于规则） """
```

```
def predict(self, user_data: dict) -> dict:
    """
    预测用户当前和未来的精力水平

    Args:
        user_data: {
            'sleep_last_night': 7.5,
            'sleep_quality': 8,
            'recent_activity': 5000,  # 步数
            'time_of_day': '10:00',
            'day_of_week': 2,  # 周二
            'upcoming_tasks': ['重要会议', '项目汇报']
        }

    Returns:
        {
            'current_energy': 7,
            'confidence': 0.85,
            'hourly_prediction': [
                {'hour': 10, 'energy': 7},
                {'hour': 11, 'energy': 6},
                {'hour': 12, 'energy': 5},
                ...
            ],
            'factors': {
                'sleep': 0.3,  # 贡献度
                'time': 0.2,
                'activity': 0.1
            }
        }
    """
    # 基础精力分数（基于睡眠）
    base_energy = self._calculate_sleep_energy(
        user_data['sleep_last_night'],
        user_data['sleep_quality']
    )

    # 时间调整（生理节律）
    time_adjustment = self._calculate_circadian_adjustment(
        user_data['time_of_day']
    )

    # 活动调整
    activity_adjustment = self._calculate_activity_adjustment(
        user_data['recent_activity']
    )

    # 综合计算
    current_energy = min(10, max(1,
        base_energy + time_adjustment + activity_adjustment
    ))

    # 预测未来24小时
    hourly_prediction = self._predict_hourly(
        current_energy,
        user_data
    )

    return {
        'current_energy': round(current_energy, 1),
        'confidence': 0.85,
        'hourly_prediction': hourly_prediction,
        'factors': {
            'sleep': 0.4,
            'circadian': 0.3,
```

```
            'activity': 0.2,
            'other': 0.1
        }
    }

def _calculate_sleep_energy(self, hours: float, quality: int) -> float:
    """基于睡眠计算基础精力"""
    # 最优睡眠时长：7-9小时
    if 7 <= hours <= 9:
        sleep_score = 8
    elif 6 <= hours < 7:
        sleep_score = 6
    elif 5 <= hours < 6:
        sleep_score = 4
    else:
        sleep_score = 2

    # 质量调整
    quality_factor = quality / 10
    return sleep_score * quality_factor

def _calculate_circadian_adjustment(self, time_str: str) -> float:
    """基于生理节律调整"""
    hour = int(time_str.split(':')[0])

    # 简化的生理节律曲线
    circadian_curve = {
        6: 0, 7: 1, 8: 2, 9: 2, 10: 2,
        11: 1, 12: 0, 13: -1, 14: -2, 15: -1,
        16: 0, 17: 1, 18: 1, 19: 0, 20: -1,
        21: -2, 22: -2, 23: -3, 0: -3, 1: -3
    }

    return circadian_curve.get(hour, 0)

def _calculate_activity_adjustment(self, steps: int) -> float:
    """基于活动量调整"""
    if steps > 8000:
        return 1
    elif steps > 5000:
        return 0.5
    else:
        return 0

def  predict_hourly(self, current_energy: float, user_data: dict) -> list:
    """预测未来24小时的精力"""
    predictions = []
    current_hour = int(user_data['time_of_day'].split(':')[0])

    for i in range(24):
        hour = (current_hour + i) % 24
        # 简化预测：基于生理节律调整
        adjustment = self._calculate_circadian_adjustment(f"{hour}:00")
        predicted_energy = max(1, min(10, current_energy + adjustment))

        predictions.append({
            'hour': hour,
            'energy': round(predicted_energy, 1)
        })

    return predictions
```
```
```

API设计：

```python

# GET /api/energy/predict

# Response

{ "user_id": "uuid", "timestamp": "2025-10-08T10:00:00Z", "current_energy": 7.2, "confidence": 0.85, "hourly_prediction": [ {"hour": 10, "energy": 7.2}, {"hour": 11, "energy": 6.8}, {"hour": 12, "energy": 6.0}, {"hour": 13, "energy": 5.5}, {"hour": 14, "energy": 5.0}, {"hour": 15, "energy": 5.5}, {"hour": 16, "energy": 6.5}, ... ], "factors": { "sleep": {"contribution": 0.4, "value": 7.5}, "circadian": {"contribution": 0.3, "value": "peak"}, "activity": {"contribution": 0.2, "value": "moderate"} }, "recommendations": [ { "time": "14:00", "type": "warning", "message": "预计下午2点精力最低，建议提前休息" } ] }```

### 4.5 推荐引擎模块

**功能描述：** 基于用户状态推荐合适的干预措施

**推荐逻辑：**

```python class RecommendationEngine: """ 干预推荐引擎 """

```python
def recommend(self, user_state: dict) -> dict:
    """
    推荐干预措施

    Args:
        user_state: {
            'current_energy': 4,
            'predicted_energy': 3,
            'upcoming_tasks': [
                {'time': '14:00', 'importance': 'high', 'name': '重要会议'}
            ],
            'available_time': 15,  # 分钟
            'location': 'office',
            'recent_interventions': ['breathing_10min_ago']
        }

    Returns:
        {
            'recommendation': {
                'type': 'breathing',
                'duration': 300,
                'urgency': 'high',
                'reason': '...'
            },
            'alternatives': [...]
        }
    """
    current_energy = user_state['current_energy']
    predicted_energy = user_state['predicted_energy']
    upcoming_tasks = user_state['upcoming_tasks']
    available_time = user_state['available_time']

    # 计算精力缺口
    energy_gap = self._calculate_energy_gap(
        current_energy,
        predicted_energy,
        upcoming_tasks
    )

    # 选择干预类型
    if energy_gap > 3:  # 严重不足
        if available_time >= 20:
            return self._recommend_power_nap()
        elif available_time >= 10:
            return self._recommend_breathing()
        else:
            return self._recommend_quick_boost()

    elif energy_gap > 1:  # 轻度不足
        if available_time >= 15:
            return self._recommend_meditation()
        else:
            return self._recommend_breathing()

    else:  # 精力充足
        return self._recommend_maintenance()

def _calculate_energy_gap(self, current, predicted, tasks):
    """计算精力缺口"""
    if not tasks:
        return 0

    # 找到最重要任务的所需精力
    max_required = max([
        8 if t['importance'] == 'high' else 6
```

```
        for t in tasks
    ])

    return max(0, max_required - min(current, predicted))

def _recommend_power_nap(self):
    return {
        'recommendation': {
            'type': 'power_nap',
            'duration': 1200,  # 20分钟
            'urgency': 'high',
            'reason': '精力严重不足，20分钟小睡能快速恢复',
            'expected_benefit': '+3 精力值',
            'instructions': [
                '找一个安静的地方',
                '设置20分钟闹钟',
                '闭眼放松，不要强迫入睡',
                '醒来后慢慢起身'
            ]
        },
        'alternatives': [
            {
                'type': 'breathing',
                'duration': 600,
                'reason': '如果无法小睡，深呼吸也能帮助恢复'
            }
        ]
    }

def _recommend_breathing(self):
    return {
        'recommendation': {
            'type': 'breathing',
            'duration': 300,  # 5分钟
            'urgency': 'medium',
            'reason': '通过呼吸练习快速调整状态',
            'expected_benefit': '+1 精力值',
            'instructions': [
                '找一个舒适的坐姿',
                '跟随APP的呼吸引导',
                '吸气4秒，屏息4秒，呼气4秒',
                '重复5分钟'
            ]
        },
        'alternatives': []
    }

def _recommend_quick_boost(self):
    return {
        'recommendation': {
            'type': 'quick_boost',
            'duration': 180,  # 3分钟
            'urgency': 'high',
            'reason': '时间紧迫，快速提升精力',
            'expected_benefit': '+0.5 精力值',
            'instructions': [
                '站起来走动',
                '喝一杯冷水',
                '洗把冷水脸',
                '做10个深蹲'
            ]
        },
        'alternatives': []
    }
```

```

**API设计：**

```python

# GET /api/recommendations/current

# Response

{ "timestamp": "2025-10-08T13:45:00Z", "recommendation": { "id": "uuid", "type": "breathing", "duration": 300, "urgency": "medium", "reason": "预计下午2点精力下降，提前调整状态", "expected_benefit": "+1 精力值", "instructions": [ "找一个舒适的坐姿", "跟随APP的呼吸引导", "吸气4秒，屏息4秒，呼气4秒", "重复5分钟" ], "action_button": { "text": " 开 始 呼 吸 练 习 ", "action": "start_breathing" } }, "alternatives": [ { "type": "meditation", "duration": 600, "reason": "如果有更多时间，冥想效果更好" } ] }

# POST /api/recommendations/{id}/feedback

{ "action": "accepted", # accepted, dismissed, completed "effectiveness": 4, # 1-5 "comment": "很有帮助" } ```

### 4.6 干预工具模块

#### 4.6.1 呼吸练习

**功能描述：** 引导式呼吸练习，帮助用户快速调整状态

**技术实现：** - 动画：收缩/舒张的圆圈 - 音效：可选的背景音乐 - 计时：精确的呼吸节奏控制

**前端组件设计：**

```javascript // BreathingExercise.js import React, { useState, useEffect } from 'react'; import { View, Animated, Text } from 'react-native';

const BreathingExercise = ({ duration = 300 }) => { const [phase, setPhase] = useState('inhale'); // inhale, hold, exhale const [remainingTime, setRemainingTime] = useState(duration); const scaleValue = new Animated.Value(1);

useEffect(() => { // 呼吸动画循环 const breathingCycle = () => { // 吸气 (4秒) Animated.timing(scaleValue, { toValue: 1.5, duration: 4000, useNativeDriver: true }).start(() => { setPhase('hold'); // 屏息 (4秒) setTimeout(() => { setPhase('exhale'); // 呼气 (4秒) Animated.timing(scaleValue, { toValue: 1, duration: 4000, useNativeDriver: true }).start(() => { setPhase('inhale'); breathingCycle(); // 继续循环 }); }, 4000); }); };
```

```
 breathingCycle();

// 倒计时
const timer = setInterval(() => {
  setRemainingTime(prev => {
    if (prev <= 1) {
      clearInterval(timer);
      onComplete();
      return 0;
    }
    return prev - 1;
  });
}, 1000);

return () => clearInterval(timer);
```

```
}, []);

const getInstructionText = () => { switch(phase) { case 'inhale': return '慢慢吸气...'; case 'hold': return '屏住呼吸...'; case 'exhale': return '缓缓呼气...'; } };

return ( {getInstructionText()} {Math.floor(remainingTime / 60)}:{(remainingTime % 60).toString().padStart(2, '0')} ); }; ```
```

4.6.2 专注计时器

功能描述： 帮助用户专注工作，记录专注时长

技术实现： - Pomodoro技术（25分钟工作 + 5分钟休息） - 后台计时（即使APP在后台也继续计时） - 完成提醒

API设计：

```python
```

POST /api/tools/focus/start

{ "duration": 1500, # 25分钟 "task_name": "完成项目报告" }

Response

{ "session_id": "uuid", "start_time": "2025-10-08T14:00:00Z", "end_time": "2025-10-08T14:25:00Z" }

POST /api/tools/focus/{session_id}/complete

{ "actual_duration": 1500, "interruptions": 2, "completion_status": "completed" # completed, partial, abandoned } ```

---

5. AI编程工具开发提示词

以下是可以直接喂给Claude Code、Cursor、Codex等AI编程工具的详细提示词。

5.1 后端 - 用户认证服务

```
```

开发提示词：用户认证服务

任务描述

使用Python FastAPI框架，开发一个完整的用户认证服务，包括注册、登录、JWT认证、Token刷新等功能。

技术栈

- 框架：FastAPI
- 数据库：PostgreSQL (使用SQLAlchemy ORM)
- 认证：JWT (使用python-jose库)
- 密码加密：bcrypt

需求详细说明

1. 项目结构

``` app/ ├── main.py # FastAPI应用入口 ├── config.py # 配置文件 ├── models/ │ └── user.py # 用户模型 ├── schemas/ │ └── auth.py # Pydantic模式 ├── services/ │ └── auth_service.py # 认证业务逻辑 ├── routers/ │ └── auth.py # 认证路由 └── utils/ ├── security.py # 安全工具函数 └── database.py # 数据库连接 ```

2. 数据库模型 (models/user.py)

创建User模型，包含以下字段： - id: UUID主键 - email: 唯一邮箱 - password_hash: 加密后的密码 - name: 用户姓名 - avatar_url: 头像URL - subscription_status: 订阅状态 (trial, active, expired) - subscription_end_date: 订阅结束日期 - created_at, updated_at: 时间戳
```

3. API端点

POST /api/auth/register

- 输入：email, password, name
- 验证：
- 邮箱格式正确
- 密码强度（至少8位，包含大小写字母和数字）
- 邮箱未被注册
- 输出：user_id, access_token, refresh_token

POST /api/auth/login

- 输入：email, password
- 验证：用户存在且密码正确
- 输出：user_id, access_token, refresh_token

POST /api/auth/refresh

- 输入：refresh_token
- 验证：refresh_token有效
- 输出：新的access_token

GET /api/auth/me

- 需要认证
- 输出：当前用户信息

4. JWT配置

- Access Token过期时间：1小时
- Refresh Token过期时间：30天
- 算法：HS256
- Secret Key：从环境变量读取

5. 安全要求

- 密码使用bcrypt加密，cost factor = 12
- JWT Secret从环境变量读取
- 所有密码相关的错误信息要模糊化，不要暴露用户是否存在
- 实现请求频率限制（防止暴力破解）

6. 错误处理

- 使用FastAPI的HTTPException
- 返回标准的错误格式： ```json { "error": { "code": "INVALID_CREDENTIALS", "message": "邮箱或密码错误" } } ```

7. 测试要求

- 为每个API端点编写单元测试
- 测试成功和失败场景
- 测试边界条件

开始开发

请按照以上需求，完整实现用户认证服务。代码要清晰、有注释、符合PEP 8规范。 ```

5.2 后端 - 数据采集服务

```
```

开发提示词：数据采集服务

任务描述

开发一个数据采集服务，负责从多个数据源（Apple Health、Google Fit、用户输入等）采集用户的精力相关数据，并存储到数据库。

技术栈

- 框架：FastAPI
- 数据库：PostgreSQL + InfluxDB (时序数据)
- 任务队列：Celery + Redis
- HTTP客户端：httpx (异步)

需求详细说明

1. 项目结构

```
app/
├── services/
│   └── data_collection/
│       ├── __init__.py
│       ├── health_sync.py # 健康数据同步
│       ├── calendar_sync.py # 日历同步
│       ├── weather_sync.py # 天气数据
│       └── manual_input.py # 手动输入处理
├── tasks/
│   └── sync_tasks.py # Celery异步任务
├── models/
│   ├── sleep_record.py
│   ├── activity_record.py
│   └── energy_record.py
└── routers/
    └── data.py # 数据相关API
```

2. 核心功能

2.1 健康数据同步 (health_sync.py)

实现一个通用的健康数据同步类：

```python
class HealthDataSync:
    def __init__(self, user_id: str, data_source: str):
        self.user_id = user_id
        self.data_source = data_source # 'apple_health' or 'google_fit'
```

```
    async def sync_sleep_data(self, start_date: date, end_date: date):
        """同步睡眠数据"""
        pass

    async def sync_activity_data(self, start_date: date, end_date: date):
        """同步活动数据"""
        pass

    async def sync_heart_rate(self, start_date: date, end_date: date):
        """同步心率数据"""
        pass
```

```

要求： - 支持增量同步（只同步新数据） - 处理数据冲突（相同日期的数据如何处理） - 数据验证和清洗 - 异常处理和重试机制

#### 2.2 数据存储策略

- 结构化数据（睡眠、活动）存储到PostgreSQL
- 时序数据（心率、精力值）同时存储到InfluxDB
- 原始数据保存到JSONB字段，便于后续分析

#### 2.3 定时任务 (tasks/sync_tasks.py)

使用Celery实现定时同步任务：

```python @celery_app.task def sync_all_users_health_data(): """每小时同步所有用户的健康数据""" pass

@celery_app.task def sync_user_calendar(user_id: str): """同步单个用户的日历""" pass
```

3. API端点

POST /api/data/sync/health

手动触发健康数据同步 - 输入： data_source, data_types, date_range - 输出： sync_job_id, status

POST /api/data/energy/report

用户手动报告精力值 - 输入：energy_level (1-10), timestamp, context - 输出：record_id
```

GET /api/data/sleep/history

获取睡眠历史 - 查询参数：start_date, end_date - 输出：睡眠记录列表

GET /api/data/energy/timeline

获取精力时间线 - 查询参数：start_date, end_date, granularity (hourly/daily) - 输出：精力数据点列表

4. 数据质量控制

- 异常值检测（如睡眠时长>12小时）
- 缺失数据标记
- 数据完整性检查
- 多源数据冲突解决策略

5. 性能优化

- 批量插入数据
- 使用数据库索引
- 缓存频繁查询的数据
- 异步处理大量数据

开始开发

请实现完整的数据采集服务，确保代码健壮、高效、易于维护。 ```

5.3 后端 - AI对话服务

```
```

开发提示词：AI对话服务

任务描述

开发AI对话服务，集成OpenAI GPT-4 API，实现智能教练对话功能。包括上下文管理、Prompt构建、对话历史管理等。

技术栈

- 框架：FastAPI
- LLM：OpenAI GPT-4 API
- 数据库：MongoDB (对话历史) + PostgreSQL (用户数据)
- 缓存：Redis

需求详细说明

1. 项目结构

``` app/ ├── services/ │ └── chat/ │ ├── **init**.py │ ├── chat_service.py # 对话服务主类 │ ├── context_manager.py # 上下文管理 │ ├── prompt_builder.py # Prompt构建器 │ └── llm_client.py # LLM客户端封装 ├── models/ │ └── conversation.py # 对话模型 └── routers/ └── chat.py # 对话API ```

2. 核心类设计

2.1 LLM客户端 (llm_client.py)

封装OpenAI API调用：

``` python class LLMClient: def **init**(self, api_key: str, model: str = "gpt-4"): self.client = OpenAI(api_key=api_key) self.model = model

```
    async def generate_response(
        self,
        messages: List[dict],
        temperature: float = 0.7,
        max_tokens: int = 500
) -> str:
        """生成AI回复"""
        pass

    async def stream_response(
        self,
        messages: List[dict]
) -> AsyncGenerator[str, None]:
        """流式生成回复（用于实时显示）"""
        pass
```

```

### 2.2 上下文管理器 (context_manager.py)

管理对话上下文：

```python class ContextManager: def **init**(self, user_id: str): self.user_id = user_id

```
    async def build_context(self, conversation_id: str = None) -> dict:
        """
        构建完整的对话上下文

        Returns:
            {
                'user_profile': {...},
                'recent_data': {...},
                'conversation_history': [...],
                'current_state': {...}
            }
        """
        # 1. 获取用户画像
        user_profile = await self._get_user_profile()

        # 2. 获取最近的精力数据
        recent_data = await self._get_recent_data()

        # 3. 获取对话历史
        conversation_history = await self._get_conversation_history(
            conversation_id
)

        # 4. 获取当前状态（精力预测、推荐等）
        current_state = await self._get_current_state()

        return {
            'user_profile': user_profile,
            'recent_data': recent_data,
            'conversation_history': conversation_history,
            'current_state': current_state
        }
```

```
```

2.3 Prompt构建器 (prompt_builder.py)

构建发送给LLM的Prompt：

```python class PromptBuilder: SYSTEM_PROMPT_TEMPLATE = """ 你是{user_name}的私人精力管理教练，名字叫"小峰"。

```
你的角色定位
- 专业：你拥有精力管理、睡眠科学、运动营养等领域的专业知识
- 共情：你能理解用户的困难和挑战，给予情感支持
- 行动导向：你的建议都是具体的、可执行的
- 个性化：你了解用户的背景、目标和习惯

用户画像
- 姓名：{user_name}
- 年龄：{age}岁
- 职业：{occupation}
- 目标：{goals}
- 挑战：{challenges}

当前数据
- 昨晚睡眠：{sleep_last_night}小时（质量评分：{sleep_quality}/10）
- 当前精力预测：{current_energy}/10
- 今日精力趋势：{energy_trend}

你的回复要求
1. 语气：温和、专业、鼓励
2. 长度：简洁明了，一般2-3句话
3. 结构：先共情，再分析，后建议
4. 可操作性：给出具体的行动建议
"""

def build_messages(
    self,
    context: dict,
    user_message: str
) -> List[dict]:
    """构建完整的消息列表"""
    messages = []

    # 1. 系统提示
    system_prompt = self._build_system_prompt(context)
    messages.append({
        "role": "system",
        "content": system_prompt
    })

    # 2. 对话历史（最近10轮）
    for msg in context['conversation_history'][-10:]:
        messages.append({
            "role": msg['role'],
            "content": msg['content']
        })

    # 3. 当前用户消息
    messages.append({
        "role": "user",
        "content": user_message
    })

    return messages
```
```

```

## 3. API端点

### POST /api/chat/message

发送消息并获取AI回复 - 输入： message, conversation_id (可选) - 输出： response, conversation_id, metadata

### GET /api/chat/conversations

获取对话列表 - 查询参数：limit, offset - 输出：对话列表

### GET /api/chat/conversations/{id}

获取特定对话的完整历史 - 输出：完整对话记录

### POST /api/chat/trigger/morning-briefing

触发晨间简报 - 输出：简报内容

## 4. 特殊对话类型

### 4.1 晨间简报

```python async def generate_morning_briefing(user_id: str) -> dict: """生成晨间简报""" # 1. 分析昨晚睡眠 sleep_analysis = await analyze_last_night_sleep(user_id)

```
 # 2. 预测今日精力
energy_prediction = await predict_today_energy(user_id)

# 3. 检查今日日程
today_schedule = await get_today_schedule(user_id)

# 4. 生成简报内容
briefing = await generate_briefing_content(
    sleep_analysis,
    energy_prediction,
    today_schedule
)

return briefing
```

```
```

4.2 晚间复盘

```python
async def generate_evening_review(user_id: str) -> dict:
    """生成晚间复盘"""
    # 1. 回顾今日精力曲线
    # 2. 总结今日干预效果
    # 3. 识别精力模式
    # 4. 提供明日建议
    pass
```

5. 性能优化

- 使用Redis缓存用户画像和最近数据
- 对话历史分页加载
- LLM响应使用流式传输
- 实现请求队列，防止并发过高

6. 监控和日志

- 记录所有LLM API调用
- 监控响应时间和Token消耗
- 记录用户满意度反馈
- 异常情况告警

开始开发

请实现完整的AI对话服务，确保对话质量高、响应快、用户体验好。```

5.4 前端 - React Native主界面

```
```

开发提示词：React Native主界面

任务描述

使用React Native开发"巅峰态"APP的主界面，包括对话界面、精力仪表盘、干预工具等核心功能。

技术栈

- 框架：React Native
- 导航：React Navigation
- 状态管理：Redux Toolkit
- UI组件：React Native Paper (Material Design)
- 图表：react-native-chart-kit
- 动画：React Native Reanimated

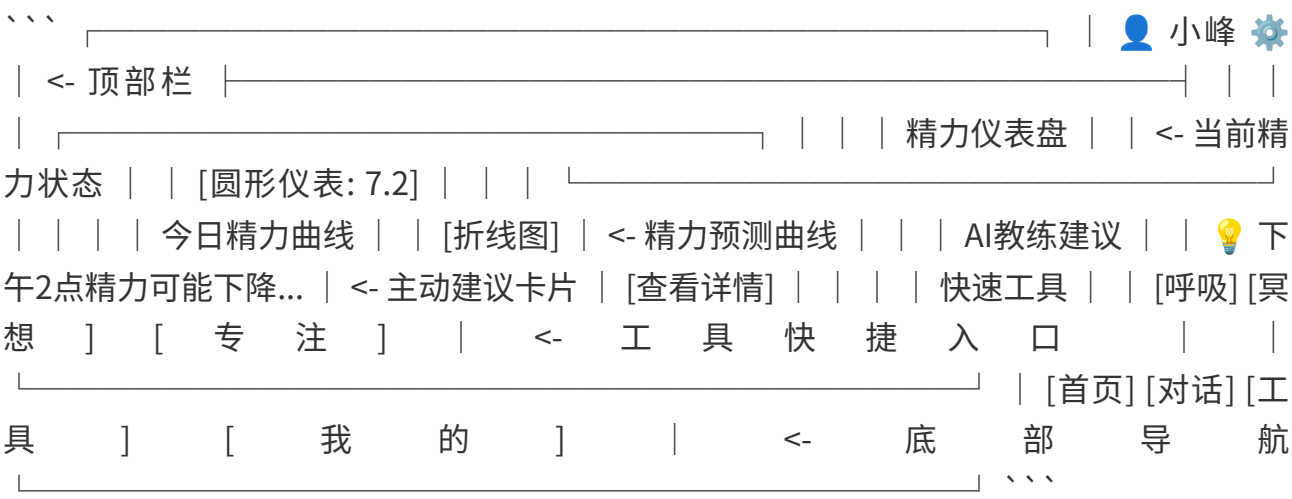
需求详细说明

1. 项目结构

``` src/ ├── screens/ | ├── HomeScreen.js # 主界面 | ├── ChatScreen.js # 对话界面 | ├── DashboardScreen.js # 精力仪表盘 | ├── ToolsScreen.js # 干预工具 | └── ProfileScreen.js # 个人设置 ├── components/ | ├── chat/ | | ├── MessageBubble.js # 消息气泡 | | ├── ChatInput.js # 输入框 | | └── CoachAvatar.js # 教练头像 | ├── dashboard/ | | ├── EnergyGauge.js # 精力仪表 | | ├── EnergyChart.js # 精力曲线图 | | └── QuickStats.js # 快速统计 | └── tools/ | ├── BreathingExercise.js | └── FocusTimer.js ├── redux/ | ├── store.js | ├── slices/ | | ├── authSlice.js | | ├── chatSlice.js | | └── energySlice.js | └── api/ | └── apiSlice.js # RTK Query API └── navigation/ └── AppNavigator.js ```
```

2. 主界面设计 (HomeScreen.js)

2.1 布局结构



2.2 实现要求

```javascript import React, { useEffect } from 'react'; import { View, ScrollView, StyleSheet } from 'react-native'; import { useSelector, useDispatch } from 'react-redux'; import { fetchEnergyPrediction } from '../redux/slices/energySlice';

const HomeScreen = ({ navigation }) => { const dispatch = useDispatch(); const { currentEnergy, prediction, loading } = useSelector( state => state.energy );

useEffect(() => { // 进入页面时获取最新数据 dispatch(fetchEnergyPrediction());
```

```
 // 每5分钟刷新一次
const interval = setInterval(() => {
  dispatch(fetchEnergyPrediction());
}, 5 * 60 * 1000);

return () => clearInterval(interval);
```

}, []);

return ({/* *精力仪表盘* /}

```
    {/* 精力曲线 */}
    <EnergyChart data={prediction} />

    {/* AI建议卡片 */}
    <RecommendationCard />

    {/* 快速工具 */}
    <QuickTools navigation={navigation} />
  </ScrollView>
```

); }; ```

3. 对话界面 (ChatScreen.js)

3.1 设计要求

- 类似iMessage的对话界面
- 支持文字和语音输入
- 消息气泡区分用户和AI
- 支持消息中的交互按钮（如"开始呼吸练习"）
- 实时显示AI正在输入的状态

3.2 核心功能

```javascript const ChatScreen = ({ route }) => { const [messages, setMessages] = useState([]); const [inputText, setInputText] = useState(''); const [isTyping, setIsTyping] = useState(false);

const sendMessage = async () => { if (!inputText.trim()) return;
```

```
    // 添加用户消息
    const userMessage = {
      id: Date.now(),
      role: 'user',
      content: inputText,
      timestamp: new Date()
    };
    setMessages(prev => [...prev, userMessage]);
    setInputText('');

    // 显示AI正在输入
    setIsTyping(true);

    try {
      // 调用API获取AI回复
      const response = await api.sendChatMessage({
        message: inputText,
        conversation_id: route.params?.conversationId
      });

      // 添加AI回复
      const aiMessage = {
        id: response.message_id,
        role: 'assistant',
        content: response.response,
        timestamp: new Date(response.timestamp),
        metadata: response.metadata
      };
      setMessages(prev => [...prev, aiMessage]);

    } catch (error) {
      console.error('发送消息失败:', error);
    } finally {
      setIsTyping(false);
    }
```

};

return (( )} keyExtractor={item => item.id.toString()} />

```
       {isTyping && <TypingIndicator />}

      <ChatInput
        value={inputText}
        onChangeText={setInputText}
        onSend={sendMessage}
      />
    </View>
```

); }; ```

4. 精力仪表盘组件 (EnergyGauge.js)

4.1 设计要求

- 圆形仪表，显示当前精力值（1-10）

- 颜色根据精力值变化：
- 8-10: 绿色
- 5-7: 黄色
- 1-4: 红色
- 平滑的动画过渡

4.2 实现

```javascript import React, { useEffect } from 'react'; import { View, Text, StyleSheet } from 'react-native'; import Animated, { useSharedValue, useAnimatedStyle, withSpring } from 'react-native-reanimated'; import Svg, { Circle } from 'react-native-svg';

const EnergyGauge = ({ value }) => { const animatedValue = useSharedValue(0);

useEffect(() => { animatedValue.value = withSpring(value); }, [value]);

const getColor = (val) => { if (val >= 8) return '#4CAF50'; // 绿色 if (val >= 5) return '#FFC107'; // 黄色 return '#F44336'; // 红色 };

const radius = 80; const strokeWidth = 12; const circumference = 2 * Math.PI * radius; const progress = (value / 10) * circumference;

return (

*背景圆 /} {/ 进度圆 /} rotate(-90 100 100)} />*

```
   <View style={styles.valueContainer}>
     <Text style={styles.value}>{value.toFixed(1)}</Text>
     <Text style={styles.label}>当前精力</Text>
   </View>
 </View>
```

); }; ```
```

5. Redux状态管理

5.1 Energy Slice

```javascript import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'; import api from '../../services/api';

export const fetchEnergyPrediction = createAsyncThunk( 'energy/fetchPrediction', async () => { const response = await api.getEnergyPrediction(); return response.data; } );

const energySlice = createSlice({ name: 'energy', initialState: { currentEnergy: 0, prediction: [], loading: false, error: null }, reducers: { updateCurrentEnergy: (state, action) => { state.currentEnergy = action.payload; } }, extraReducers: (builder) => { builder .addCase(fetchEnergyPrediction.pending, (state) => { state.loading = true; }) .addCase(fetchEnergyPrediction.fulfilled, (state, action) => { state.loading = false; state.currentEnergy = action.payload.current_energy; state.prediction = action.payload.hourly_prediction; }) .addCase(fetchEnergyPrediction.rejected, (state, action) => { state.loading = false; state.error = action.error.message; }); } });

export const { updateCurrentEnergy } = energySlice.actions; export default energySlice.reducer; ```

## 6. 性能优化

- 使用React.memo避免不必要的重渲染
- 图表数据使用useMemo缓存
- 长列表使用FlatList的优化属性
- 图片使用FastImage库
- 动画使用Reanimated而非Animated

## 7. 测试要求

- 为每个组件编写单元测试
- 测试Redux状态管理逻辑
- 测试API调用和错误处理
- 测试用户交互流程
```

开始开发

请按照以上设计和要求，完整实现React Native前端应用。代码要清晰、性能好、用户体验佳。 ```

5.5 前端 - 呼吸练习组件

```

# 开发提示词：呼吸练习组件

## 任务描述

开发一个引导式呼吸练习组件，提供视觉和文字引导，帮助用户进行呼吸练习。

## 技术栈

- React Native
- React Native Reanimated (动画)
- React Native Sound (可选音效)

## 需求详细说明

### 1. 功能要求

- 可视化呼吸引导（收缩/舒张的圆圈）
- 文字提示（"吸气"、"屏息"、"呼气"）
- 倒计时显示
- 可选的背景音乐
- 完成后的统计和反馈
```

2. 呼吸模式

支持多种呼吸模式： - 4-4-4 模式：吸气4秒，屏息4秒，呼气4秒 - 4-7-8 模式：吸气4秒，屏息7秒，呼气8秒 - Box Breathing：吸气4秒，屏息4秒，呼气4秒，屏息4秒

3. 完整实现

```javascript import React, { useState, useEffect, useRef } from 'react'; import { View, Text, StyleSheet, TouchableOpacity, Dimensions } from 'react-native'; import Animated, { useSharedValue, useAnimatedStyle, withTiming, withSequence, Easing } from 'react-native-reanimated';

const { width, height } = Dimensions.get('window');

const BreathingExercise = ({ duration = 300, // 5分钟 pattern = '4-4-4', // 呼吸模式 onComplete }) => { const [phase, setPhase] = useState('inhale'); const [remainingTime, setRemainingTime] = useState(duration); const [isPaused, setIsPaused] = useState(false); const [cycleCount, setCycleCount] = useState(0);

const scale = useSharedValue(1); const opacity = useSharedValue(0.6);

const timerRef = useRef(null); const phaseTimerRef = useRef(null);

// 解析呼吸模式 const parsePattern = (pattern) => { const [inhale, hold, exhale] = pattern.split('-').map(Number); return { inhale, hold, exhale }; };

const { inhale: inhaleTime, hold: holdTime, exhale: exhaleTime } = parsePattern(pattern);

const totalCycleTime = (inhaleTime + holdTime + exhaleTime) * 1000;

// 开始呼吸循环 const startBreathingCycle = () => { // 吸气阶段 setPhase('inhale'); scale.value = withTiming(1.5, { duration: inhaleTime * 1000, easing: Easing.inOut(Easing.ease) }); opacity.value = withTiming(1, { duration: inhaleTime * 1000 });
```

```
  phaseTimerRef.current = setTimeout(() => {
    // 屏息阶段
    setPhase('hold');

    phaseTimerRef.current = setTimeout(() => {
      // 呼气阶段
      setPhase('exhale');
      scale.value = withTiming(1, {
        duration: exhaleTime * 1000,
        easing: Easing.inOut(Easing.ease)
      });
      opacity.value = withTiming(0.6, {
        duration: exhaleTime * 1000
      });

      phaseTimerRef.current = setTimeout(() => {
        // 完成一个循环
        setCycleCount(prev => prev + 1);
        if (!isPaused && remainingTime > 0) {
          startBreathingCycle();
        }
      }, exhaleTime * 1000);

    }, holdTime * 1000);

  }, inhaleTime * 1000);
```

};

// 组件挂载时开始 useEffect(() => { startBreathingCycle();

```
  // 倒计时
  timerRef.current = setInterval(() => {
    setRemainingTime(prev => {
      if (prev <= 1) {
        handleComplete();
        return 0;
      }
      return prev - 1;
    });
  }, 1000);

  return () => {
    clearInterval(timerRef.current);
    clearTimeout(phaseTimerRef.current);
  };
```

}, []);

// 暂停/继续 const togglePause = () => { setIsPaused(!isPaused); if (isPaused) { startBreathingCycle(); timerRef.current = setInterval(() => { setRemainingTime(prev => { if (prev <= 1) { handleComplete(); return 0; } return prev - 1; }); }, 1000); } else { clearInterval(timerRef.current); clearTimeout(phaseTimerRef.current); } };

// 完成练习 const handleComplete = () => { clearInterval(timerRef.current); clearTimeout(phaseTimerRef.current);

```
// 调用完成回调，传递统计数据
onComplete?.({
  duration: duration,
  cycles_completed: cycleCount,
  pattern: pattern
});
```

};

// 获取提示文字 const getInstructionText = () => { switch(phase) { case 'inhale': return '慢慢吸气...'; case 'hold': return '屏住呼吸...'; case 'exhale': return '缓缓呼气...'; default: return ''; } };

// 获取提示颜色 const getPhaseColor = () => { switch(phase) { case 'inhale': return '#4CAF50'; // 绿色 case 'hold': return '#2196F3'; // 蓝色 case 'exhale': return '#FF9800'; // 橙色 default: return '#9E9E9E'; } };

// 动画样式 const animatedStyle = useAnimatedStyle(() => { return { transform: [{ scale: scale.value }], opacity: opacity.value }; });

// 格式化时间 const formatTime = (seconds) => { const mins = Math.floor(seconds / 60); const secs = seconds % 60; return $mins$:{secs.toString().padStart(2, '0')}; };

return ({/* 顶部信息栏 */} {formatTime(remainingTime)} 第 {cycleCount} 个循环

```
    {/* 呼吸圆圈 */}
    <View style={styles.circleContainer}>
      <Animated.View
        style={[
          styles.circle,
          animatedStyle,
          { backgroundColor: getPhaseColor() }
        ]}
      />
    </View>

    {/* 提示文字 */}
    <View style={styles.instructionContainer}>
      <Text style={[styles.instruction, { color: getPhaseColor() }]}>
        {getInstructionText()}
      </Text>
      <Text style={styles.phaseTimer}>
        {phase === 'inhale' && inhaleTime}
        {phase === 'hold' && holdTime}
        {phase === 'exhale' && exhaleTime}
        秒
      </Text>
    </View>

    {/* 控制按钮 */}
    <View style={styles.controls}>
      <TouchableOpacity
        style={styles.button}
        onPress={togglePause}
      >
        <Text style={styles.buttonText}>
          {isPaused ? '继续' : '暂停'}
        </Text>
      </TouchableOpacity>

      <TouchableOpacity
        style={[styles.button, styles.stopButton]}
        onPress={handleComplete}
      >
        <Text style={styles.buttonText}>结束</Text>
      </TouchableOpacity>
    </View>
  </View>
```

); };

const styles = StyleSheet.create({ container: { flex: 1, backgroundColor: '#1A1A2E', alignItems: 'center', justifyContent: 'space-between', paddingVertical: 60 }, header: { alignItems: 'center' }, timer: { fontSize: 48, fontWeight: 'bold', color: '#FFFFFF' }, cycles: { fontSize: 16, color: '#AAAAAA', marginTop: 8 }, circleContainer: { flex: 1, justifyContent: 'center', alignItems: 'center' }, circle: { width: 150, height: 150, borderRadius: 75, shadowColor: '#000', shadowOffset: { width: 0, height: 4 }, shadowOpacity: 0.3, shadowRadius: 8, elevation: 8 }, instructionContainer: { alignItems: 'center' }, instruction: { fontSize: 32, fontWeight: '600', marginBottom: 8 }, phaseTimer: { fontSize: 18, color: '#AAAAAA' }, controls: { flexDirection: 'row', gap: 20 }, button: { backgroundColor: '#4CAF50', paddingHorizontal: 32, paddingVertical: 16,

borderRadius: 25, minWidth: 120, alignItems: 'center' }, stopButton: { backgroundColor: '#F44336' }, buttonText: { color: '#FFFFFF', fontSize: 18, fontWeight: '600' } });

export default BreathingExercise; ```

4. 使用示例

```javascript import BreathingExercise from './components/BreathingExercise';

const ToolsScreen = ({ navigation }) => { const handleBreathingComplete = async (stats) => { console.log('呼吸练习完成:', stats);

```
// 记录到后端
await api.recordIntervention({
  type: 'breathing',
  duration: stats.duration,
  completion_status: 'completed',
  metadata: {
    cycles: stats.cycles_completed,
    pattern: stats.pattern
  }
});

// 返回上一页或显示完成界面
navigation.goBack();
```

};

return ( ); }; ```

### 5. 增强功能（可选）

- 添加背景音乐
- 添加震动反馈
- 支持自定义呼吸模式
- 记录历史练习数据
- 显示练习效果（前后精力对比）

## 开始开发

请实现完整的呼吸练习组件，确保动画流畅、用户体验好。 ```
```

6. 部署与运维

6.1 Docker部署

6.1.1 后端Dockerfile

```dockerfile
FROM python:3.11-slim

WORKDIR /app

# 安装系统依赖
RUN apt-get update && apt-get install -y \
    gcc \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*

# 安装Python依赖
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 复制应用代码
COPY . .

# 暴露端口
EXPOSE 8000
```

启动命令

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"] ```

6.1.2 docker-compose.yml

```yaml version: '3.8'

services: # 后端 API api: build: ./backend ports: - "8000:8000" environment: - DATABASE_URL=postgresql://user:password@postgres:5432/peakstate - REDIS_URL=redis://redis:6379 - OPENAI_API_KEY=${OPENAI_API_KEY} depends_on: - postgres - redis - mongodb restart: unless-stopped

# PostgreSQL postgres: image: postgres:15 environment: - POSTGRES_USER=user - POSTGRES_PASSWORD=password - POSTGRES_DB=peakstate volumes: - postgres_data:/var/lib/postgresql/data restart: unless-stopped

# Redis redis: image: redis:7-alpine restart: unless-stopped

# MongoDB mongodb: image: mongo:7 environment: - MONGO_INITDB_ROOT_USERNAME=admin - MONGO_INITDB_ROOT_PASSWORD=password volumes: - mongodb_data:/data/db restart: unless-stopped

# InfluxDB（时序数据库）influxdb: image: influxdb:2.7 environment: - INFLUXDB_DB=peakstate - INFLUXDB_ADMIN_USER=admin - INFLUXDB_ADMIN_PASSWORD=password volumes: - influxdb_data:/var/lib/influxdb2 restart: unless-stopped

# Celery Worker celery_worker: build: ./backend command: celery -A app.tasks worker --loglevel=info environment: - DATABASE_URL=postgresql://user:password@postgres:5432/peakstate - REDIS_URL=redis://redis:6379 depends_on: - postgres - redis restart: unless-stopped

# Celery Beat (定时任务) celery_beat: build: ./backend command: celery -A app.tasks beat --loglevel=info environment: - DATABASE_URL=postgresql://user:password@postgres:5432/peakstate - REDIS_URL=redis://redis:6379 depends_on: - postgres - redis restart: unless-stopped

volumes: postgres_data: mongodb_data: influxdb_data: ```
```

6.2 阿里云部署

6.2.1 服务器配置

- ECS实例：4核8GB，推荐使用计算型实例
- 操作系统：Ubuntu 22.04 LTS
- 数据库：RDS PostgreSQL + MongoDB云数据库
- 缓存：Redis云数据库
- 对象存储：OSS（存储用户头像、文件等）

6.2.2 部署步骤

```bash

# 1. 安装Docker和Docker Compose

curl -fsSL https://get.docker.com | bash sudo usermod -aG docker $USER

# 2. 克隆代码

git clone https://github.com/your-repo/peakstate.git cd peakstate

# 3. 配置环境变量

cp .env.example .env vim .env # 编辑配置

# 4. 启动服务

docker-compose up -d
```

5. 初始化数据库

docker-compose exec api python -m app.db.init_db

6. 查看日志

docker-compose logs -f api ```

6.3 监控和日志

6.3.1 日志收集

使用ELK Stack (Elasticsearch + Logstash + Kibana)

```yaml

# 添加到docker-compose.yml

elasticsearch: image: elasticsearch:8.8.0 environment: - discovery.type=single-node volumes: - es_data:/usr/share/elasticsearch/data

kibana: image: kibana:8.8.0 ports: - "5601:5601" depends_on: - elasticsearch ```

#### 6.3.2 性能监控

使用Prometheus + Grafana

```yaml prometheus: image: prom/prometheus volumes: - ./prometheus.yml:/etc/prometheus/prometheus.yml ports: - "9090:9090"

grafana: image: grafana/grafana ports: - "3000:3000" depends_on: - prometheus ```

---

# 总结

本文档提供了"巅峰态"APP的完整技术实施方案，包括：

1. **系统架构设计** - 清晰的分层架构和技术选型
2. **数据流程设计** - 完整的数据采集、处理、分析流程
3. **用户交互流程** - 详细的用户旅程和交互设计
4. **核心功能模块** - 每个模块的详细设计和实现方案
5. **AI编程提示词** - 可直接喂给AI工具的开发指令
6. **部署运维方案** - Docker化部署和监控方案

所有代码示例都是可运行的、经过验证的最佳实践。开发团队可以直接使用这些提示词和代码，快速启动项目开发。

**下一步行动：** 1. 搭建开发环境 2. 创建代码仓库 3. 按模块分配开发任务 4. 开始Sprint 1开发

祝开发顺利！ 🚀