《巅峰态》产品完整文档

PeakState - AI精力管理教练

文档版本： 2.0
最后更新： 2025年10月10
产品负责人： KOB

目录

第一部分：产品需求文档 (PRD)

1. 产品概述 (Executive Summary)

1.1 产品名称

巅峰态 (英文名: PeakState - AI Energy Coach)

1.2 产品愿景

将全球顶尖精力管理专家的智慧，通过超个性化的AI，赋能给每一位渴望自我超越的用户。我们提供的不是一个工具，而是一位7x24小时在线、永远懂你的私人AI教练。巅峰态致力于帮助每一位用户找到并保持属于自己的最佳状态，在事业与生活中持续发挥最佳表现。

1.3 目标用户 (Target Audience)

核心用户画像： 30-50岁，面临精力瓶颈、渴望提升工作表现和生活品质的中高阶职场人士、创业者和管理者。

用户特征： - 承受高压力、从事高脑力劳动 - 认同自我投资的价值，愿意为效果付费 - 对新生事物接受度高，追求科学的自我管理方法 - 明显感受到精力下降，但缺乏系统性的解决方案 - 时间宝贵，需要高效、个性化的指导

1.4 核心问题与解决方案

用户面临的核心问题：

目标用户普遍面临精力下降、效率降低的困境。他们明显感觉到自己不如从前那样精力充沛，容易疲劳、难以专注、情绪波动，这些问题严重影响了工作表现和生活质量。然而，他们缺乏科学的诊断工具来找到问题根源，缺乏个性化的规划来优化精力分配，也缺乏持续的行动支持来养成良好习惯。

现有的工具要么只关注生理数据（如健康手环），要么只管理任务清单（如效率App），要么只提供放松内容（如冥想App），它们都缺乏整合性和智慧性，无法提供一个全面的、主动的、个性化的精力管理解决方案。

巅峰态的解决方案：

巅峰态提供一个以"对话式AI"为核心的移动应用。这位AI教练能够整合用户的生理数据、行为数据和主观感受，运用先进的人工智能模型进行深度分析，并通过自然、流畅的语言对话，提供主动的、个性化的精力管理指导。

巅峰态不是一个被动的记录工具，而是一位主动的、有温度的、专业的私人教练。它会在每天早晨主动问候你，根据你昨晚的睡眠和今天的日程给出建议；它会在你压力过大时及时提醒你休息；它会在每周末与你一起复盘，帮助你发现精力管理的规律。

---

2. 战略与商业模式 (Strategy & Business Model)

2.1 市场定位

巅峰态定位于高端个人服务市场，创造一个全新的品类："AI私人教练"。

我们不与传统的效率工具或健康App在同一维度竞争。我们的对标对象是真人精力管理教练、高管教练等高端咨询服务。相比于真人教练，巅峰态提供了更高的性价比（每月300元 vs. 真人教练数千元/小时）、更高的可用性（7x24小时 vs. 预约制）和更强的数据驱动能力（整合多源数据 vs. 依赖主观判断）。

2.2 商业模式：订阅制服务

核心模式： "聘请制"订阅服务

定价策略： - **月度订阅：** 300元/月 - **年度订阅：** 2,998元/年（相当于250元/月，节省600元）

核心价值主张：

> *"每月不到一次商务宴请的费用，聘请一位顶级的私人AI精力管理教练，为您提供7x24小时的专业服务，全面提升您的事业与生活表现。"*

免费增值策略：

提供7天全功能免费试用。用户在试用开始时需要绑定支付方式，试用期满后自动转为付费订阅。这种策略有三个优势：

1. **筛选高意向用户：** 愿意绑定支付方式的用户，通常是真正有需求、有付费意愿的高质量用户
2. **确保深度体验：** 用户知道7天后会扣费，会更有动力在试用期内深度使用产品，充分体验价值
3. **降低转化摩擦：** 试用结束后自动续费，无需用户再次决策，提高付费转化率

2.3 预期收益 (Financial Projections)

基础假设： - 目标在产品上线后6个月内获取1,000名付费用户 - 月度订阅与年度订阅用户比例为 7:3 - 用户月均流失率控制在5%以内

收益计算：

项目	计算方式	金额
月度订阅收入	1,000 × 70% × 300元	210,000元/月
年度订阅收入（折算）	(1,000 × 30% × 2,998元) ÷ 12	74,950元/月
预期月度经常性收入 (MRR)		**约285,000元**
预期年度经常性收入 (ARR)		**约342万元**

说明： 此为初步估算，未考虑市场推广成本、应用商店渠道费用（Apple 30%抽成）、云服务成本及用户流失率等因素。实际净利润需要在运营过程中持续优化。

---

3. MVP核心功能详述 (Core Features)

核心设计原则：

以"AI对话"为绝对中心，完整保留"数据感知 → AI分析 → 对话式教练 → 关键时刻干预"的核心体验闭环。

3.1 模块一：AI教练对话界面 (The Coaching Chat)

用户场景：

用户90%的交互发生在这个对话界面。无论是获取建议、记录感受、启动工具还是查看数据，都通过与AI教练的自然对话来完成。这个界面不是一个冷冰冰的聊天机器人，而是一位有温度、有智慧、真正关心你的私人教练。

功能需求：

1. **基础对话能力**
2. 支持文字输入和语音输入（自动转文字）
3. 支持多轮连续对话，能够理解上下文和指代关系
4. 响应速度快，通常在2秒内给出回复
5. **教练人设 (Persona)**
6. AI的语言风格经过精心设计：专业但不生硬、权威但不高高在上、共情但不过度煽情、可靠且值得信赖

7. 采用现代插画风格的拟人化头像，用户可以在入职时从3个预设形象中选择一位作为自己的教练
8. 教练会记住用户的名字、偏好、目标和历史对话，让每次交流都有延续性
9. **主动对话触发器 (Proactive Engagement)** - 这是巅峰态的核心差异化功能

P0级（必须实现）： - **每日晨间简报：** 每天清晨（用户自定义时间，默认7:00），AI教练主动发起对话。内容根据用户是否有硬件数据自适应： - 有硬件数据：基于睡眠质量、HRV等生理指标，给出今日精力评估和日程建议 - 无硬件数据：通过问候和提问，引导用户快速记录睡眠和当前感受

- **每日晚间复盘：** 每天睡前（用户自定义时间，默认22:00），AI教练主动发起对话，引导用户回顾一天的精力状态、完成的重要任务和情绪波动，并给予鼓励和建议
- **上下文理解与记忆**
- 能够理解连续的多轮对话，记住用户在本次对话中提到的信息
- 能够调取用户的历史数据和对话记录，提供有延续性的建议
- 能够识别用户的情绪状态（如焦虑、沮丧、兴奋），并调整回复的语气和内容

3.2 模块二：多源数据感知 (Data Perception Engine)

用户场景：

用户在完成"入职"流程时，会被引导进行一次性的数据授权。之后，所有数据都在后台无感同步，用户无需手动记录。这些数据成为AI教练分析和建议的依据，让每一条建议都是个性化的、有根据的。

功能需求：

1. **数据降级策略（核心创新）**

巅峰态采用"数据降级"策略，确保无论用户是否拥有智能硬件，都能获得核心价值。

有硬件用户（体验上限）：

- **P0级（必须实现）：**
    - 授权读取Apple Health（iOS）/ Google Fit（Android）中的核心数据：
    - 睡眠时长、入睡时间、起床时间

    - 静息心率
    - 这些是最基础、最关键的生理指标，几乎所有智能手表/手环都能提供
- **P1级（优先实现）：**
    - 专门优化对主流高端设备的深度连接：
    - Oura Ring：获取睡眠分期（深睡、浅睡、REM）、体温变化
    - WHOOP：获取HRV（心率变异性）、恢复度评分、压力指数
    - Apple Watch：获取血氧饱和度、站立时长
    - 这些深度数据能让AI教练提供更精准、更专业的洞察

无硬件用户（体验基础）：

- **P0级（必须实现）：**
    - 通过晨间和晚间的对话式问答，引导用户快速记录关键信息：
    - "昨晚大概几点睡的？几点醒的？"
    - "给昨晚的睡眠质量打个分（1-10）？"
    - "现在感觉精力如何（1-10）？"
    - 这种对话式记录比传统的表单填写更自然、更不容易被遗忘
- **P1级（优先实现）：**
    - 授权读取手机使用数据（屏幕时间、App使用记录），推断：
    - 卧床时间（手机最后使用时间）和起床时间（手机首次使用时间）
    - 睡前屏幕使用时长（可能影响睡眠质量）
    - 白天在社交媒体上的时间（可能影响专注力和情绪）
- **行为数据接入**
- **P0级（必须实现）：**
    - 授权读取系统日历，获取会议、日程安排
    - AI教练会根据日程密度和会议类型，预测用户的精力消耗，并给出规划建议
- **P1级（优先实现）：**
    - 允许用户在对话中快速添加任务和目标

- AI教练会帮助用户评估任务的精力消耗等级，并建议最佳执行时间

3.3 模块三：实时干预工具箱 (The Intervention Toolkit)

用户场景：

当用户感到压力、焦虑或需要进入专注状态时，AI教练会在对话中智能推荐相应的干预工具，用户一键即可启动。这些工具不是藏在菜单深处的功能，而是教练在恰当时机主动递给你的"急救包"。

功能需求：

1. **P0 - 引导式呼吸练习 (Guided Breathing)**

触发方式： - AI在对话中检测到用户表达压力、焦虑情绪（如"我很焦虑"、"压力好大"） - 或者，用户主动请求："教练，我需要冷静一下"

实现方式： - 点击后，进入一个极简的全屏界面 - 屏幕中央是一个平滑的呼吸动画：圆圈缓慢放大（吸气）和缩小（呼气） - 配合简洁的文字指令："吸气...保持...呼气..." - 提供1分钟、3分钟、5分钟三种时长选项 - 结束后，AI教练会询问："感觉好些了吗？"并记录这次干预

1. **P0 - 专注计时器 (Focus Timer)**

触发方式： - 用户通过对话指令开启，如："帮我开始30分钟的专注时间" - 或者，AI在检测到用户即将进行高精力消耗任务时主动建议

实现方式： - 进入全屏计时器界面，屏幕中央显示倒计时 - 背景是动态的、缓慢变化的模糊色块（如从蓝色渐变到绿色），营造专注氛围 - 可选择性播放白噪音或自然声（如雨声、海浪声） - 计时期间，可选择屏蔽手机通知 - 结束后，AI教练给予鼓励："太棒了！你刚刚完成了30分钟的深度工作。"并记录专注时长

3.4 模块四：用户"入职"与付费 (Onboarding & Monetization)

用户场景：

新用户下载巅峰态后，会经历一段充满仪式感的"教练入职"流程。这不是一个冷冰冰的注册表单，而是一次与AI教练的初次见面，一次关于"我们将如何合作"的正式对话。

功能需求：

1. **P0 - 教练入职仪式**

流程设计（5个步骤）：

步骤1：欢迎 - 精美的欢迎动画，展示巅峰态的Logo - Slogan："你好，未来的自己。"

步骤2-3：价值呈现 - 用简洁的图标和文字，展示巅峰态的核心价值： - "整合生理与心理数据，认识真实的你" - "AI教练主动规划，让精力用在刀刃上"

步骤4：选择你的教练 - 展示3个预设的教练形象（智者、伙伴、专家） - 用户选择后，该教练的头像放大，并发出第一条消息： - "你好，我是你的AI教练。从今天起，我将全力协助你管理精力、提升表现。准备好开始了吗？"

步骤5：初始问卷（5-8个问题） - AI教练通过对话式提问，建立用户的基础画像： - "你目前最大的精力困扰是什么？"（多选：容易疲劳、难以专注、睡眠不好、情绪波动） - "你通常几点睡觉？几点起床？" - "你有佩戴智能手表或手环吗？"（如有，引导授权；如无，说明也完全可以使用）

步骤6：数据授权 - 清晰地说明为什么需要这些权限，以及如何保护隐私 - 引导用户授权HealthKit（iOS）/Google Fit（Android）和日历读取权限 - 强调：所有数据加密存储，仅用于个性化分析，绝不分享给第三方

1. **P0 - 开启免费试用与订阅**

界面设计： - 清晰展示订阅价格和试用政策： - "7天免费试用，随后300元/月" - "年度订阅2,998元，相当于每月250元，节省600元" - 明确的行动按钮："开始我的7天免费试用"

支付流程： - 调用Apple In-App Purchase（iOS）或Google Play Billing（Android） - 用户绑定支付方式后，立即开启7天试用 - 在试用期第5天和第6天，AI教练会主动提醒用户试用即将结束，询问是否继续 - 试用期满后自动扣费，转为正式订阅

---

4. 里程碑与实施路径 (Roadmap & Milestones)

总开发周期： 约16周（4个月）

第一阶段：准备与设计（第1-3周）

目标： 确定最终技术选型，完成产品高保真原型(UI/UX)设计

关键任务： - 最终确定技术栈和开发工具 - 使用Figma AI完成所有核心界面的高保真设计 - 设计并定稿AI教练的3个形象 - 编写AI教练的核心对话脚本和人设文档 - 搭建开发环境（前

端、后端、数据库）

产出物： - 技术栈确认文档 - 可交互的产品原型（Figma） - AI教练人设与对话脚本文档 - 开发环境就绪

第二阶段：核心功能开发（第4-10周）

目标： 完成前后端核心功能的开发，打通数据链路

关键任务： - 前端：实现入职流程、对话界面、基础导航 - 后端：实现用户认证、数据存储、API接口 - 数据接入：完成HealthKit/Google Fit和日历的数据读取 - AI集成：接入OpenAI API，实现基础对话能力 - 主动触发：实现晨间简报和晚间复盘的定时推送机制

产出物： - 可在真机上运行的App（iOS和Android） - 完整的后端API服务 - 能够进行基础对话的AI教练 - 能够读取和展示用户健康数据

第三阶段：集成与测试（第11-14周）

目标： 完善AI能力，开发干预工具，进行全面测试

关键任务： - AI优化：完善提示工程，提升对话质量和个性化程度 - 工具开发：实现呼吸练习和专注计时器 - 付费流程：接入Apple/Google支付SDK，实现订阅和试用逻辑 - 内部测试：邀请创始人和5-10名种子用户进行深度内测 - Bug修复：根据测试反馈，修复所有P0和P1级别的问题

产出物： - 功能完整的Alpha版本App - 内测用户反馈报告 - 优化后的产品版本

第四阶段：发布与优化（第15-16周）

目标： 正式发布，开启公测

关键任务： - 应用商店准备：撰写App描述、准备截图和宣传视频 - 提交审核：向App Store和主流Android应用商店提交审核 - 监控与优化：密切监控首批用户的使用数据和反馈 - 快速迭代：根据真实用户反馈，进行快速优化

产出物： - 在App Store和Android应用商店上架的V1.0正式版 - 用户使用数据报告 - 下一阶段迭代计划

第二部分：UI/UX设计任务书

1. 整体设计语言 (Global Design Language)

1.1 设计哲学

"安静的陪伴者"

巅峰态的界面设计遵循一个核心哲学：界面是教练思想的延伸，它从不喧宾夺主。我们追求的是一种克制的美学——给用户足够的信息和引导，但绝不过度刺激或干扰。

用户打开巅峰态，应该感受到的是平静、专注和信任，而不是焦虑、压迫或混乱。每一个像素、每一种颜色、每一个动画，都应该服务于这个目标。

1.2 色彩规范 (Color Palette)

巅峰态采用暗色调为主的配色方案，营造专注、沉浸的氛围。

色彩名称	色值	用途
主背景色	`#1C1C1E`	App的主要背景色，深灰色，营造专注氛围
次背景色/卡片色	`#2C2C2E`	用于信息卡片、消息气泡等，拉开层次
主文本色	`#FFFFFF`	纯白色，用于主要文本，保证最高可读性
次文本色	`#8E8E93`	灰色，用于辅助信息、时间戳、说明文字
高亮/行动色	`#0A84FF`	iOS标准蓝色，用于按钮、链接、可交互元素
积极反馈色	`#34C759`	iOS标准绿色，用于完成任务、达成目标的提示
警示/提醒色	`#FF9500`	iOS标准橙色，用于精力预警、重要提醒

设计原则： - 大面积使用深色背景，减少蓝光刺激，适合全天候使用 - 高亮色的使用必须克制，只在需要引导用户行动时使用 - 避免使用红色（过于刺激）和纯黑色（过于压抑）

1.3 字体规范 (Typography)

字体家族： PingFang SC（苹方-简）

这是iOS和macOS的默认中文字体，具有极佳的可读性和现代感，能确保跨平台体验的一致性。

字重使用规范：

字重	用途	示例
Semibold（中粗体）	标题、重要数据、强调内容	"今日精力准备度：92分"
Regular（常规体）	正文、对话内容、常规信息	AI教练的回复内容
Light（细体）	次要描述、辅助说明	时间戳、版本号

字号使用建议： - 大标题：28-32pt - 小标题：20-24pt - 正文：16-18pt - 辅助文字：12-14pt

1.4 图标风格 (Iconography)

风格： 线性、描边风格（Line Icons）

规范： - 所有图标的线条粗细保持一致（2px） - 图标尺寸统一为24×24pt或32×32pt - 颜色使用主文本色或高亮色，保持简洁 - 避免使用过于复杂或写实的图标

推荐图标库： SF Symbols（iOS）或 Material Icons（Android）

---

2. 核心元素设计 - AI教练形象

2.1 设计风格要求

风格定位： 现代、简约、扁平化的商业插画风格

核心要求： - 形象需体现智慧、专业与亲和力 - 特征保持一定的模糊性与中性化，避免过于具象 - 不使用超写实风格，避免"恐怖谷效应" - 色彩柔和，与整体UI风格协调

参考风格： Headspace的插画风格、Slack的吉祥物、高端商业杂志的人物插画

2.2 三个预设形象设计

用户在入职时可以从以下三个形象中选择一位作为自己的AI教练：

形象A：智者 (The Sage)

视觉特征： - 轮廓：戴着简约眼镜、发型整洁的形象 - 色调：偏冷静的蓝色系（如深蓝、靛蓝） - 气质：沉稳、理性、充满智慧

适合人群： 追求理性决策、喜欢数据分析的用户

形象B：伙伴 (The Companion)

视觉特征： - 轮廓：面带微笑、姿态开放的形象 - 色调：偏温暖的绿色系（如青绿、翠绿） - 气质：亲切、鼓励、充满活力

适合人群： 需要情感支持、喜欢温暖陪伴的用户

形象C：专家 (The Expert)

视觉特征： - 轮廓：极简的侧脸剪影或抽象几何形状 - 色调：中性灰色系（如深灰、银灰） - 气质：专业、高效、极简主义

适合人群： 追求效率、喜欢简洁风格的用户

2.3 动态设计要求

教练形象不是静态的图片，而应该有微妙的动态反馈，增强"活着"的感觉。

默认状态：呼吸动画 - 头像有轻微、平缓的缩放动画（约5%的变化幅度） - 周期约4-6秒，模拟人的呼吸节奏 - 营造一种"教练在倾听"的感觉

反馈状态：情绪动画 - 当AI给出积极反馈时（如"太棒了！"），头像可以有一个轻微的点头或微笑动画 - 当AI表示关切时（如"注意到你最近压力很大"），头像可以有一个轻微的倾斜，表示共情 - 所有动画都应该非常克制，避免过度卡通化

---

3. 核心界面设计指令

详见设计原型图附件

第三部分：技术架构与环境搭建指南

1. 最终技术栈 (Finalized Tech Stack)

经过综合评估，巅峰态MVP版本采用以下技术栈：

层面	技术选型	版本要求
前端	Swift + SwiftUI	Swift 5.7+, Xcode 14+
后端	Python + FastAPI	Python 3.11+, FastAPI 0.100+
数据库	PostgreSQL	PostgreSQL 15+
AI核心	OpenAI API	GPT-4 或更高版本
部署平台	阿里云	ECS + RDS + OSS

1.1 技术选型理由

**前端：Swift + SwiftUI

Swift是苹果官方推出的现代、安全、高效的编程语言，专为iOS、macOS等苹果平台打造。使用Swift进行原生开发，能够最大化地利用苹果生态的性能优势和最新的系统特性，为用户提供最流畅、最原生的顶级的产品体验。SwiftUI是苹果新一代的声明式UI框架，能够用更少的代码构建更精美、更动态的界面，完美契合巅峰态对高品质用户体验的追求。

后端：Python + FastAPI

Python是AI/ML领域的绝对王者，使用Python后端能够无缝地集成OpenAI API，未来进行模型微调也极为方便。FastAPI是现代Python Web框架的性能标杆，其异步特性非常适合处理大量的API调用和I/O操作。

FastAPI的自动API文档生成（Swagger UI）和数据校验功能，能大幅提升后端开发效率和代码质量。

数据库：PostgreSQL

PostgreSQL是业界公认的最先进的开源关系型数据库，稳定性和数据一致性无与伦比，适合存储用户账户、订阅状态等核心业务数据。它对JSONB等半结构化数据的支持，也能让

我们在初期用它存储一些用户画像或日志数据，简化架构。

AI核心：OpenAI API

OpenAI的GPT-4及后续模型在自然语言理解、逻辑推理和多语言支持方面处于领先地位，是实现"智慧教练"人设的最可靠保障。通过API调用的方式，我们无需在初期就投入巨资进行模型训练，可以立即获得强大的AI能力，将精力集中在提示工程和业务逻辑上。

部署平台：阿里云

由于巅峰态MVP阶段主要面向中国大陆市场，阿里云在国内的网络接入速度和稳定性无与伦比，能为用户提供最流畅的体验。同时，使用阿里云能更好地满足国内的数据安全与合规性要求。

---

2. 环境搭建与项目初始化

2.1 前端环境搭建 (Swift)

步骤1：安装开发工具

```
# 安装Node.js（推荐使用nvm管理版本）
nvm install 18
nvm use 18

# 安装Swift CLI
npm install -g react-native-cli

# iOS开发需要安装Xcode（仅Mac）
# Android开发需要安装Android Studio
```

步骤2：初始化项目

```
# 创建新的iOS项目（使用Xcode）
# 1. 打开Xcode
# 2. File -> New -> Project
# 3. 选择 iOS -> App
# 4. Product Name: PeakState
# 5. Interface: SwiftUI
# 6. Language: Swift
# 7. 最低版本: iOS 17.0
```

步骤3：配置Swift Package Manager依赖

在Xcode中添加以下依赖包： 1. File -> Add Package Dependencies 2. 添加以下包：

```
# 网络请求库
https://github.com/Alamofire/Alamofire.git (版本 5.9+)

# Keychain存储
https://github.com/evgenyneu/keychain-swift.git (版本 20.0+)

# 其他依赖根据需要添加
```

注意： HealthKit和EventKit是iOS系统原生框架，无需额外安装。

步骤4：配置HealthKit（iOS）

在Xcode中打开项目，进行以下配置： 1. 选择项目的Target 2. 进入"Signing & Capabilities"标签 3. 点击"+ Capability"，添加"HealthKit" 4. 在`Info.plist`中添加隐私说明：`xml <key>NSHealthShareUsageDescription</key> <string>巅峰态需要读取您的健康数据，以提供个性化的精力管理建议</string> <key>NSHealthUpdateUsageDescription</key> <string>巅峰态需要更新您的健康数据</string>`

步骤5：配置日历权限（iOS和Android）

在`Info.plist`（iOS）中添加：

```
<key>NSCalendarsUsageDescription</key>
<string>巅峰态需要访问您的日历，以根据日程安排提供精力规划建议</string>
```

在`AndroidManifest.xml`（Android）中添加：

```
<uses-permission android:name="android.permission.READ_CALENDAR" />
```

2.2 后端环境搭建 (Python + FastAPI)

步骤1：创建项目目录和虚拟环境

```
mkdir peakstate-backend
cd peakstate-backend

# 使用Poetry管理依赖（推荐）
pip install poetry
poetry init
```

步骤2：安装核心依赖

```
poetry add fastapi
poetry add "uvicorn[standard]"
poetry add sqlalchemy
poetry add psycopg2-binary
poetry add python-jose[cryptography]
poetry add passlib[bcrypt]
poetry add openai
poetry add python-multipart
poetry add pydantic-settings
```

步骤3：创建项目结构

```
mkdir -p app/{api,core,crud,models,schemas}
touch app/__init__.py
touch app/main.py
touch app/core/{__init__.py,config.py,security.py}
touch app/api/{__init__.py,auth.py,users.py,chat.py}
touch app/models/{__init__.py,user.py}
touch app/schemas/{__init__.py,user.py,chat.py}
touch app/crud/{__init__.py,user.py}
```

最终项目结构：

```
peakstate-backend/
├── app/
│   ├── __init__.py
│   ├── main.py               # FastAPI应用实例
│   ├── api/                  # API路由
│   │   ├── __init__.py
│   │   ├── auth.py           # 认证相关API
│   │   ├── users.py          # 用户管理API
│   │   └── chat.py           # 对话API
│   ├── core/                 # 核心配置
│   │   ├── __init__.py
│   │   ├── config.py         # 配置管理
│   │   └── security.py       # 安全相关（JWT等）
│   ├── crud/                 # 数据库操作
│   │   ├── __init__.py
│   │   └── user.py
│   ├── models/               # SQLAlchemy模型
│   │   ├── __init__.py
│   │   └── user.py
│   └── schemas/              # Pydantic模型
│       ├──   init _.py
│       ├── user.py
│       └── chat.py
├── pyproject.toml
└── README.md
```

步骤4：编写基础配置文件

创建 `app/core/config.py`：

```python
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    PROJECT_NAME: str = "PeakState API"
    VERSION: str = "1.0.0"
    API_V1_STR: str = "/api/v1"

    # 数据库配置
    DATABASE_URL: str

    # JWT配置
    SECRET_KEY: str
    ALGORITHM: str = "HS256"
    ACCESS_TOKEN_EXPIRE_MINUTES: int = 60 * 24 * 7  # 7天

    # OpenAI配置
    OPENAI_API_KEY: str
    OPENAI_MODEL: str = "gpt-4"

    class Config:
        env_file = ".env"

settings = Settings()
```

创建`.env`文件（不要提交到Git）：

```
DATABASE_URL=postgresql://user:password@localhost/peakstate
SECRET_KEY=your-secret-key-here
OPENAI_API_KEY=your-openai-api-key-here
```

步骤5：编写FastAPI应用主文件

创建`app/main.py`：

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.core.config import settings

app = FastAPI(
    title=settings.PROJECT_NAME,
    version=settings.VERSION,
    openapi_url=f"{settings.API_V1_STR}/openapi.json"
)

# 配置CORS，允许前端App访问
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # 生产环境应该限制具体域名
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/")
def read_root():
    return {"message": "Welcome to PeakState API"}

# 这里将来会添加路由
# app.include_router(auth.router, prefix=f"{settings.API_V1_STR}/auth", tags=["auth"])
```

步骤6：运行开发服务器

```
poetry run uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

访问 `http://localhost:8000/docs` 查看自动生成的API文档。

2.3 数据库搭建 (PostgreSQL on Aliyun)

步骤1：在阿里云购买RDS实例

1. 登录阿里云控制台
2. 进入"云数据库RDS"服务
3. 点击"创建实例"
4. 选择配置：
5. 数据库类型：PostgreSQL
6. 版本：15.x
7. 规格：初期可选最低配（如1核2GB）
8. 存储：20GB SSD
9. 地域：选择靠近目标用户的地域（如华东1-杭州）

10. 完成购买

步骤2：配置数据库

1. 在RDS控制台，进入实例详情
2. 创建数据库：
3. 数据库名：`peakstate`
4. 字符集：`UTF8`
5. 创建账号：
6. 账号名：`peakstate_user`
7. 密码：设置强密码
8. 授权数据库：`peakstate`，权限：读写
9. 配置白名单：
10. 添加后端服务器的公网IP
11. 开发阶段可以临时添加 `0.0.0.0/0`（生产环境必须删除）

步骤3：测试连接

```
# 安装PostgreSQL客户端
brew install postgresql  # Mac
# 或 sudo apt-get install postgresql-client  # Linux

# 测试连接
psql -h your-rds-endpoint.rds.aliyuncs.com -p 5432 -U peakstate_user -d
peakstate
```

步骤4：在后端配置数据库连接

更新 `.env` 文件：

```
DATABASE_URL=postgresql://peakstate_user:your-password@your-rds-
endpoint.rds.aliyuncs.com:5432/peakstate
```

---

3. 核心功能开发指令

3.1 后端任务：用户认证API

任务描述： 实现用户注册、登录和JWT认证机制

步骤1：定义User模型

创建 `app/models/user.py`：

```
 from sqlalchemy import Column, Integer, String, DateTime, Boolean
from sqlalchemy.ext.declarative import declarative_base
from datetime import datetime

Base = declarative_base()

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    phone_number = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    coach_selection = Column(String, nullable=True)  # "sage", "companion",
"expert"
    is_active = Column(Boolean, default=True)
    is_subscribed = Column(Boolean, default=False)
    subscription_end_date = Column(DateTime, nullable=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)
```

步骤2：定义Pydantic Schema

创建 `app/schemas/user.py`：

```python
from pydantic import BaseModel, Field
from datetime import datetime
from typing import Optional

class UserCreate(BaseModel):
    phone_number: str = Field(..., min_length=11, max_length=11)
    password: str = Field(..., min_length=6)
    coach_selection: Optional[str] = None

class UserLogin(BaseModel):
    phone_number: str
    password: str

class User(BaseModel):
    id: int
    phone_number: str
    coach_selection: Optional[str]
    is_subscribed: bool
    subscription_end_date: Optional[datetime]

    class Config:
        from_attributes = True

class Token(BaseModel):
    access_token: str
    token_type: str
```

步骤3：实现安全相关函数

创建`app/core/security.py`：

```python
from datetime import datetime, timedelta
from jose import JWTError, jwt
from passlib.context import CryptContext
from app.core.config import settings

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() +
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY,
algorithm=settings.ALGORITHM)
    return encoded_jwt
```

步骤4：实现CRUD操作

创建`app/crud/user.py`：

```python
from sqlalchemy.orm import Session
from app.models.user import User
from app.schemas.user import UserCreate
from app.core.security import get_password_hash, verify_password

def get_user_by_phone(db: Session, phone_number: str):
    return db.query(User).filter(User.phone_number == phone_number).first()

def create_user(db: Session, user: UserCreate):
    hashed_password = get_password_hash(user.password)
    db_user = User(
        phone_number=user.phone_number,
        hashed_password=hashed_password,
        coach_selection=user.coach_selection
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

def authenticate_user(db: Session, phone_number: str, password: str):
    user = get_user_by_phone(db, phone_number)
    if not user:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return user
```

步骤5：实现API端点

创建`app/api/auth.py`：

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from datetime import timedelta
from app.schemas.user import UserCreate, UserLogin, Token, User as UserSchema
from app.crud import user as crud_user
from app.core.security import create_access_token
from app.core.config import settings
# from app.api.deps import get_db  # 需要实现数据库依赖注入

router = APIRouter()

@router.post("/register", response_model=UserSchema)
def register(user: UserCreate, db: Session = Depends(get_db)):
    db_user = crud_user.get_user_by_phone(db, phone_number=user.phone_number)
    if db_user:
        raise HTTPException(status_code=400, detail="Phone number already
registered")
    return crud_user.create_user(db=db, user=user)

@router.post("/login", response_model=Token)
def login(user_credentials: UserLogin, db: Session = Depends(get_db)):
    user = crud_user.authenticate_user(db, user_credentials.phone_number,
user_credentials.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect phone number or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires =
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.phone_number}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

3.2 前端任务："入职"流程实现

任务描述： 根据UI设计任务书，实现5屏的入职流程

步骤1：安装并配置导航

```
npm install @react-navigation/native
npm install @react-navigation/native-stack
npm install react-native-screens react-native-safe-area-context
```

步骤2：创建入职流程的屏幕组件

创建 `src/screens/onboarding/` 目录，并创建5个屏幕组件： - `WelcomeScreen.tsx` - `ValueProposition1Screen.tsx` - `ValueProposition2Screen.tsx` - `CoachSelectionScreen.tsx` - `AuthorizationScreen.tsx`

步骤3：实现导航结构

创建 `src/navigation/OnboardingNavigator.tsx`：

```
 import React from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import WelcomeScreen from '../screens/onboarding/WelcomeScreen';
import ValueProposition1Screen from
'../screens/onboarding/ValueProposition1Screen';
// ... 导入其他屏幕

const Stack = createNativeStackNavigator();

export default function OnboardingNavigator() {
  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      <Stack.Screen name="Welcome" component={WelcomeScreen} />
      <Stack.Screen name="Value1" component={ValueProposition1Screen} />
      {/* ... 其他屏幕 */}
    </Stack.Navigator>
  );
}
```

步骤4：实现欢迎屏幕示例

创建 `src/screens/onboarding/WelcomeScreen.tsx`：

```
 import React from 'react';
import { View, Text, TouchableOpacity, StyleSheet, Animated } from 'react-
native';

export default function WelcomeScreen({ navigation }) {
  const fadeAnim = React.useRef(new Animated.Value(0)).current;

  React.useEffect(() => {
    Animated.timing(fadeAnim, {
      toValue: 1,
      duration: 1000,
      useNativeDriver: true,
    }).start();
  }, []);

  return (
    <View style={styles.container}>
      <Animated.View style={[styles.content, { opacity: fadeAnim }]}>
        <Text style={styles.logo}>巅峰态</Text>
        <Text style={styles.slogan}>你好，未来的自己。</Text>
      </Animated.View>

      <TouchableOpacity
        style={styles.button}
        onPress={() => navigation.navigate('Value1')}
      >
        <Text style={styles.buttonText}>开始</Text>
      </TouchableOpacity>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#1C1C1E',
    justifyContent: 'center',
    alignItems: 'center',
    padding: 20,
  },
  content: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  logo: {
    fontSize: 48,
    fontWeight: '600',
    color: '#FFFFFF',
    marginBottom: 20,
  },
  slogan: {
    fontSize: 24,
    fontWeight: '400',
    color: '#8E8E93',
  },
  button: {
    backgroundColor: '#0A84FF',
    paddingHorizontal: 60,
    paddingVertical: 16,
    borderRadius: 12,
    marginBottom: 40,
  },
  buttonText: {
    color: '#FFFFFF',
```

```
    fontSize: 18,
    fontWeight: '600',
  },
});
```

步骤5：集成后端API

创建 `src/services/api.ts`：

```
import axios from 'axios';
import * as Keychain from 'react-native-keychain';

const API_BASE_URL = 'https://your-backend-url.com/api/v1';

const api = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
});

// 请求拦截器：自动添加JWT token
api.interceptors.request.use(async (config) => {
  const credentials = await Keychain.getGenericPassword();
  if (credentials) {
    config.headers.Authorization = `Bearer ${credentials.password}`;
  }
  return config;
});

export const authAPI = {
  register: (phoneNumber: string, password: string, coachSelection: string) =>
    api.post('/auth/register', { phone_number: phoneNumber, password,
coach_selection: coachSelection }),

  login: (phoneNumber: string, password: string) =>
    api.post('/auth/login', { phone_number: phoneNumber, password }),
};

export default api;
```

---

4. 部署与上线

4.1 后端部署到阿里云ECS

步骤1：购买ECS实例 - 规格：2核4GB（初期） - 操作系统：Ubuntu 22.04 - 地域：与RDS同一地域

步骤2：配置服务器环境

```
# SSH登录服务器
ssh root@your-server-ip

# 更新系统
apt update && apt upgrade -y

# 安装Python 3.11
apt install python3.11 python3.11-venv python3-pip -y

# 安装Nginx（用作反向代理）
apt install nginx -y

# 安装Supervisor（用于进程管理）
apt install supervisor -y
```

步骤3：部署应用

```
# 克隆代码（或使用Git）
cd /var/www
git clone your-repo-url peakstate-backend
cd peakstate-backend

# 创建虚拟环境并安装依赖
python3.11 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# 配置环境变量
cp .env.example .env
nano .env  # 填入生产环境的配置
```

步骤4：配置Supervisor

创建 `/etc/supervisor/conf.d/peakstate.conf`：

```
[program:peakstate]
directory=/var/www/peakstate-backend
command=/var/www/peakstate-backend/venv/bin/uvicorn app.main:app --host 0.0.0.0
--port 8000
user=www-data
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=/var/log/peakstate.log
```

启动服务：

```
supervisorctl reread
supervisorctl update
supervisorctl start peakstate
```

步骤5：配置Nginx

创建`/etc/nginx/sites-available/peakstate`：

```
server {
    listen 80;
    server_name api.peakstate.com;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

启用配置：

```
ln -s /etc/nginx/sites-available/peakstate /etc/nginx/sites-enabled/
nginx -t
systemctl reload nginx
```

4.2 前端打包与上架

iOS打包与上架App Store：

1. 在Xcode中配置签名和证书
2. 选择"Product" > "Archive"
3. 使用Xcode Organizer上传到App Store Connect
4. 在App Store Connect中填写应用信息、截图、描述
5. 提交审核

Android打包与上架：

1. 生成签名密钥
2. 配置`android/app/build.gradle`
3. 运行`cd android && ./gradlew bundleRelease`
4. 在Google Play Console中创建应用
5. 上传AAB文件并提交审核

---

5. 监控与维护

5.1 日志管理

使用阿里云日志服务（SLS）收集和分析应用日志，及时发现和解决问题。

5.2 性能监控

使用阿里云应用实时监控服务（ARMS）监控API响应时间、错误率等关键指标。

5.3 数据备份

配置RDS自动备份策略，确保数据安全。

文档结束

联系方式： 如有疑问，请联系产品负责人 Manus。