谋事 PlanWise | 项目主要模块设计文档

〇、项目背景与定位

谋事(Planwise) 是一款革新任务管理体验的智能待办事项应用，将传统待办清单与先进AI助手无缝融合。它不仅帮助用户记录任务，更能理解任务上下文，提供个性化完成建议，智能分析时间模式，并与用户建立对话式互动。

通过精心设计的分类系统、灵活的时间管理工具和直观的用户界面，谋事让规划变得既高效又愉悦。无论是处理日常琐事、管理工作项目还是协调个人目标，谋事都能成为您的得力助手，帮助您更智慧地规划时间，更从容地完成任务。

谋事，让每一件事都在掌握之中。

一、系统架构概述

1.1. 架构模式选择 MVVM

在构建谋事应用时，我们深思熟虑地选择了**MVVM**(Model-View-ViewModel)架构模式。这一选择基于以下几个关键考虑：

1. **数据驱动视图**：MVVM模式允许我们实现数据与视图的自动同步，确保用户界面始终反映最新的数据状态。这对于实时任务管理和AI交互特别重要。
2. **关注点分离**：通过将界面、业务逻辑和数据处理清晰地分层，我们显著提高了代码的可维护性和可测试性。
3. **状态管理优势**：在处理复杂的任务状态变化和AI交互时，MVVM的状态管理机制能够提供更可靠的数据流控制。

1.2. 系统架构设计

我们的系统架构设计充分考虑了现代应用开发的需求，采用模块化设计思想，将系统划分为相互独立但又紧密协作的功能模块：

```
系统整体架构
├── 前端模块（UI层和交互控制）
|    ├── 视图层（Activities/Fragments/XML布局）
|    ├── 视图模型层（ViewModels）
└── 后端模块（数据和业务逻辑）
     ├── 数据层（Room数据库/Repository/Entity）
     ├── 网络层（API服务/同步管理）
     └── AI交互层（第三方API集成）
```

这种分层架构不仅确保了各个模块的独立性，也为未来的功能扩展和维护提供了良好的基础。

二、前端模块设计

2.1 视图层-UI模块

视图层设计秉承"简约而不简单"的理念，在保证功能完整的同时，致力于提供流畅直观的用户体验。我们精心设计了五个核心UI模块：

a. 主界面模块

- **功能定位**：作为应用的中枢神经，整合各项核心功能，提供直观的任务概览和快捷操作。
- 核心组件：
  - 智能任务卡片流
  - 快速添加任务面板
  - 多维度任务统计仪表板
  - 智能提醒中心

b. 任务管理模块

- **功能定位**：提供全方位的任务操作体验，支持多样化的任务管理需求。
- 核心组件：
  - 任务创建向导
  - 详情编辑面板
  - 进度追踪时间轴
  - 智能提醒设置器

c. 分类管理模块

- **功能定位**：构建个性化的任务分类体系，实现科学的任务组织。
- 核心组件：
  - 分类树状视图
  - 标签管理系统
  - 智能分类建议
  - 自定义视图配置

d. AI交互模块

- **功能定位**：打造智能化的任务助手体验，提供个性化建议。
- 核心组件：
  - 对话式交互界面
  - 智能建议卡片
  - 任务优化分析器
  - 情境感知面板

e. 设置与用户模块

- **功能定位**：提供个性化配置选项，确保用户体验的可定制性。
- 核心组件：
    - 用户配置中心
    - 数据同步管理
    - 主题与样式设置
    - 隐私与安全控制

2.2. 视图模型层-Model模块

视图模型层作为连接用户界面和数据层的桥梁，承担着数据处理和业务逻辑的重要职责：

a. TaskViewModel

- 核心职责：
    - 维护任务生命周期
    - 处理任务状态转换
    - 管理任务依赖关系
    - 实现智能排序算法
- 关键功能：
    - 任务数据实时同步
    - 批量操作处理
    - 智能提醒触发
    - 历史记录追踪

b. CategoryViewModel

- 核心职责：
    - 构建分类体系框架
    - 维护分类关系树
    - 优化分类结构
- 关键功能：
    - 动态分类调整
    - 智能分类推荐
    - 分类数据统计
    - 跨分类任务关联

c. AIViewModel

- 核心职责：
    - 管理AI交互状态
    - 协调对话上下文
    - 处理智能建议生成

- 关键功能：
  - 上下文会话维护
  - 响应优先级管理
  - 建议质量评估
  - 用户反馈学习

d. UserViewModel

- 核心职责：
  - 用户状态管理
  - 偏好设置控制
  - 数据同步协调
- 关键功能：
  - 身份验证流程
  - 个性化配置存储
  - 云同步状态监控
  - 账户安全管理

三、后端模块

3.1 数据层模块

数据层作为应用的基石，采用多层次的数据管理架构，确保数据的可靠性和高效访问：

a. 实体模块

实体设计理念：采用领域驱动设计思想，将业务概念准确映射为数据模型。

核心实体定义：

- **TodoEntity**：
  - 基础属性：标题、描述、优先级
  - 时间属性：创建时间、截止时间、提醒时间
  - 状态属性：完成状态、进度追踪
  - 关联属性：分类ID、标签集合
- **CategoryEntity**：
  - 分类属性：名称、图标、颜色
  - 结构属性：父级ID、层级深度
  - 统计属性：任务计数、完成率
- **UserEntity**：
  - 账户信息：ID、名称、邮箱
  - 权限信息：角色、访问级别
  - 配置信息：偏好设置、通知选项
- **ChatMessageEntity**：
  - 消息属性：内容、类型、时间戳

    - 会话属性：上下文ID、连续性标记
    - 状态属性：处理状态、优先级

b. 数据访问模块

设计原则：遵循单一职责原则，为每类数据操作提供专门的访问接口。

核心组件：

- **TodoDao**：
    - 基础CRUD操作
    - 高级查询功能
    - 批量处理接口
    - 统计分析方法
- **CategoryDao**：
    - 分类树操作
    - 关系维护方法
    - 批量更新接口
    - 查询优化处理
- **UserPreferencesDao**：
    - 配置存取操作
    - 偏好同步方法
    - 缓存管理接口

c. 数据库模块

架构特点：采用Room持久化库，实现高效可靠的本地数据存储。

核心组件：

- **AppDatabase**：
    - 数据库实例管理
    - 表结构定义
    - 索引优化设计
- **DatabaseMigrations**：
    - 版本升级策略
    - 数据迁移脚本
    - 兼容性保证
- **DataTypeConverters**：
    - 类型转换规则
    - 序列化处理
    - 格式标准化

d. Repository模块

设计理念：采用仓库模式，统一管理数据访问逻辑。

核心实现：

- **TodoRepository**：
  - 本地数据操作封装
  - 远程同步策略
  - 缓存机制实现
- **CategoryRepository**：
  - 分类数据统一管理
  - 树结构维护逻辑
  - 关系完整性保证
- **UserRepository**：
  - 用户数据管理
  - 认证信息处理
  - 配置同步控制

3.2 网络层模块

网络层采用模块化设计，确保数据传输的可靠性和效率，同时提供完善的错误处理机制。

a. API服务模块

设计理念：采用RESTful架构，确保接口的规范性和可扩展性。

核心服务：

- **CloudSyncService**：
  - 增量同步机制
  - 冲突检测策略
  - 数据压缩传输
  - 断点续传支持
  - 加密通道维护
- **AuthService**：
  - OAuth2.0认证流程
  - Token管理机制
  - 会话状态维护
  - 安全策略实施
  - 多端登录控制
- **APIClient**：
  - 请求队列管理
  - 重试机制实现
  - 超时控制策略

    - 并发请求处理
    - 响应缓存优化

b. 同步模块

实现策略：采用双向同步机制，确保数据一致性。

核心组件：

- **SyncManager**：
    - 同步状态追踪
    - 队列优先级管理
    - 带宽利用优化
    - 电量感知控制
    - 网络状态适配
- **ConflictResolver**：
    - 版本控制策略
    - 差异对比算法
    - 合并规则定义
    - 用户确认机制
    - 回滚方案设计
- **OfflineQueueManager**：
    - 离线操作记录
    - 同步顺序维护
    - 存储空间管理
    - 定期清理策略
    - 重要性排序

c. 网络工具模块

功能定位：提供网络操作的基础设施支持。

核心工具：

- **NetworkStateMonitor**：
    - 网络质量检测
    - 连接状态追踪
    - 流量统计分析
    - 网络切换处理
    - 省电模式适配
- **RequestInterceptor**：
    - 请求头规范化
    - 参数验证处理
    - 日志记录追踪
    - 安全检查实施

    - 性能监控统计
- **ResponseCache**：
    - 多级缓存策略
    - 过期处理机制
    - 容量管理算法
    - 优先级控制
    - 内存优化方案

3.3 AI交互层模块

AI交互层致力于提供智能、自然的用户体验，通过深度学习技术优化任务管理流程。

a. AI服务模块

设计思路：整合多模型能力，提供场景化的智能服务。

核心组件：

- **AIServiceClient**：
    - 多模型调度
    - 服务质量监控
    - 负载均衡控制
    - 成本优化策略
    - 降级方案管理
- **PromptGenerator**：
    - 上下文理解
    - 模板动态生成
    - 个性化调整
    - 多语言支持
    - 效果优化反馈
- **ResponseParser**：
    - 结构化解析
    - 语义理解处理
    - 格式标准化
    - 质量评估机制
    - 异常处理流程

b. AI处理模块

实现目标：提供精准的任务管理建议和智能辅助功能。

核心功能：

- **SuggestionProcessor**：
    - 建议生成策略
    - 优先级评估

    - 场景适配处理
    - 用户反馈学习
    - 持续优化机制
- **ContextManager**：
    - 会话状态维护
    - 上下文关联分析
    - 历史记录追踪
    - 意图理解优化
    - 知识图谱构建
- **AIModelSelector**：
    - 模型特性匹配
    - 性能成本平衡
    - 实时调整策略
    - 效果评估机制
    - 备选方案管理

四、模块间交互

4.1 数据流向

数据流向的设计遵循单向数据流原则，确保数据流转的可预测性和可追踪性。

核心数据流路径：

```
用户界面(UI) <--> ViewModel <--> Repository <--> [本地数据库/网络API]
```

4.2 关键事件的模块交互设计

1. **任务创建与更新流程**：
    - 用户输入触发UI事件
    - ViewModel接收并验证数据
    - Repository层处理数据持久化
    - 触发本地存储和云端同步
    - 状态更新反馈至界面
2. **AI智能建议流程**：
    - 用户触发建议请求
    - ViewModel准备上下文信息
    - AI服务处理并生成建议
    - 结果经过优化和过滤
    - 最终呈现在用户界面
3. **数据同步流程**：
    - 系统检测到数据变更

- SyncManager评估同步需求
- 执行增量数据同步
- 处理潜在的冲突情况
- 确保各端数据一致性