

# 谋事 (Planwise) 项目中期文档-汇总



## 谋事 (Planwise) 项目中期文档-汇总

### ○、项目背景与定位

#### 一、需求分析

##### 1. 用户角色分析

###### 1.1 学生用户

###### 1.2 职场人士

###### 1.3 家庭管理者

##### 2. 用户故事

###### 2.1 学生用户故事

###### 2.2 职场人士用户故事

###### 2.3 家庭管理者用户故事

##### 3. 功能点清单

###### 3.1 待办事项管理

###### 3.2 待办事项查看与筛选

###### 3.3 数据同步

###### 3.4 AI智能助手

###### 3.5 扩展功能

##### 4. 需求优先级

#### 二、界面设计

##### 1. 今日待办

###### a. 新建日程

###### b. 日程详情

- 2. 日程
- 3. 我的
- 三、项目关键Activity设计与实现
  - a. MainActivity
  - b. TodayTodoFragment (今日待办)
  - c. AddScheduleActivity (新建日程)
  - d. ScheduleDetailActivity (日程详情)
  - e. CalendarFragment (日程)
  - f. ProfileFragment (我的)
  - g. 数据层实现
  - h. 架构设计
- 四、项目主要模块设计
  - 1. 系统架构概述
    - 1.1. 架构模式选择 MVVM
    - 1.2. 系统架构设计
  - 2. 前端模块设计
    - 2.1 视图层-UI模块
    - 2.2. 视图模型层-Model模块
  - 3. 后端模块
    - 3.1 数据层模块
    - 3.2 网络层模块
    - 3.3 AI交互层模块
  - 4. 模块间交互
    - 4.1 数据流向
    - 4.2 关键事件的模块交互设计
- 五、项目重难点与挑战的分析
  - 1. AI集成相关挑战
    - 1.1 AI响应延迟问题
    - 1.2 AI内容质量不稳定
  - 2. 数据存储与同步挑战
    - 2.1 数据库性能与扩展性
    - 2.2 数据安全性与隐私保护
  - 3. 开发流程与协作挑战
    - 3.1 需求变更与迭代管理
    - 3.2 跨平台兼容性问题

## 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

**谋事，让每一件事都在掌握之中。**

# 一、需求分析

---

## 1. 用户角色分析

在谋事（Planwise）应用中，主要存在以下角色：

### 1.1 学生用户

**特征：**

- 时间安排复杂，需管理课程、作业、考试和社交等活动
- 常面临截止日期压力
- 需要细致地进行代办规划和提醒

**需求：**

- 按课程科目/事件类型分类任务
- 设置截止时间避免错过截止日期
- 获取高效完成学习任务的建议

### 1.2 职场人士

**特征：**

- 需要协调多个项目和会议
- 任务优先级变动频繁

**需求：**

- 工作与个人任务分类
- 基于地点和时间的任务提醒
- 提高工作效率的专业建议

### 1.3 家庭管理者

**特征：**

- 管理家庭成员的各项活动和安排
- 协调家务、购物和家庭事务
- 需要灵活应对计划变更

**需求：**

- 简单直观的界面
- 按家庭成员或活动类型分类任务
- 日常事务管理的实用建议

## 2. 用户故事

### 2.1 学生用户故事

1. **作为一名高中生**，我想同时管理学校作业和课外活动，确保我不会错过任何截止日期或重要事件。
2. **作为一名大学生**，我希望能按课程类别/事件类型查看我的待办事项，这样我可以在复习特定科目时集中处理相关任务。
3. **作为一名研究生**，我需要为我的论文研究安排多个阶段性任务，并希望获得有关如何高效进行学术研究的建议。

### 2.2 职场人士用户故事

1. **作为一名项目经理**，我需要追踪多个项目的进度和截止日期，并希望获得有关如何优化时间分配的建议。
2. **作为一名销售代表**，我希望按客户和地点组织我的会议和跟进任务，这样我可以在特定区域高效安排多个会面。
3. **作为一名自由职业者**，我需要管理来自不同客户的多个项目，并希望获得如何平衡工作量的建议。
4. 更多职业...

### 2.3 家庭管理者用户故事

1. **作为一位家长**，我希望轻松记录和跟踪孩子的学校活动、医疗预约和课外课程。
2. **作为家庭主要采购者**，我需要管理购物清单并根据地点分类，以便高效完成多个地点的购物任务。
3. **作为家庭事务协调者**，我希望获得关于如何更高效地安排和完成家务的建议。
4. 更多家庭身份...

## 3. 功能点清单

### 3.1 待办事项管理

- 添加新的待办事项
- 删除现有待办事项
- 为待办事项添加类别标签
- 为待办事项设置时间
- 为待办事项添加地点信息
- 快速修改待办事项的完成状态

### 3.2 待办事项查看与筛选

- 按时间顺序查看未完成的待办事项
- 按类别筛选未完成的待办事项
- 筛选特定时间范围内的待办事项
- 查看全部已完成的待办事项
- 查看全部未完成的待办事项

3.3 数据同步

- 将本地待办事项数据上传至云端
- 从云端下载待办事项数据至本地
- 处理本地与云端数据冲突

3.4 AI智能助手

- 为用户的计划提供针对性的专业建议
- 可以为用户提供与AI基于这个任务建议的后续对话

3.5 扩展功能

- 待办事项提醒通知
- 待办事项优先级设置
- 周期性/重复性待办事项
- 待办事项详细备注
- 待办事项分享功能

4. 需求优先级

优先级	功能
必要实现	待办事项的添加、删除、状态修改 待办事项的类别标签、时间和地点 按时间和类别查看筛选 查看已完成/未完成待办事项
次要功能	本地与云端数据同步 高级AI分析与建议
可选功能	待办事项优先级 周期性待办事项 详细备注 分享功能

二、界面设计

1. 今日待办

用户可以通过前面的checkbox选择该日程是否已完成。

9:41



# Today 26 Dec

☐ 喝8杯水

健康

🕒 6:00 AM

---

☐ 移动应用软件开发ddl

作业

🕒 7:00 AM

---

☐ 社团活动

社交

🕒 10:00 AM

---

☐ 做瑜伽 15分钟

健康

🕒 21:30 AM

---



今日待办



日程



我的

#### a. 新建日程

在“今日待办”界面，点击右下方加号（+）即可跳转到“新建日程”界面。

9:41



# 新建日程



T 标题



📍 地点

🕒 全天



开始

3月1日

01:30

结束

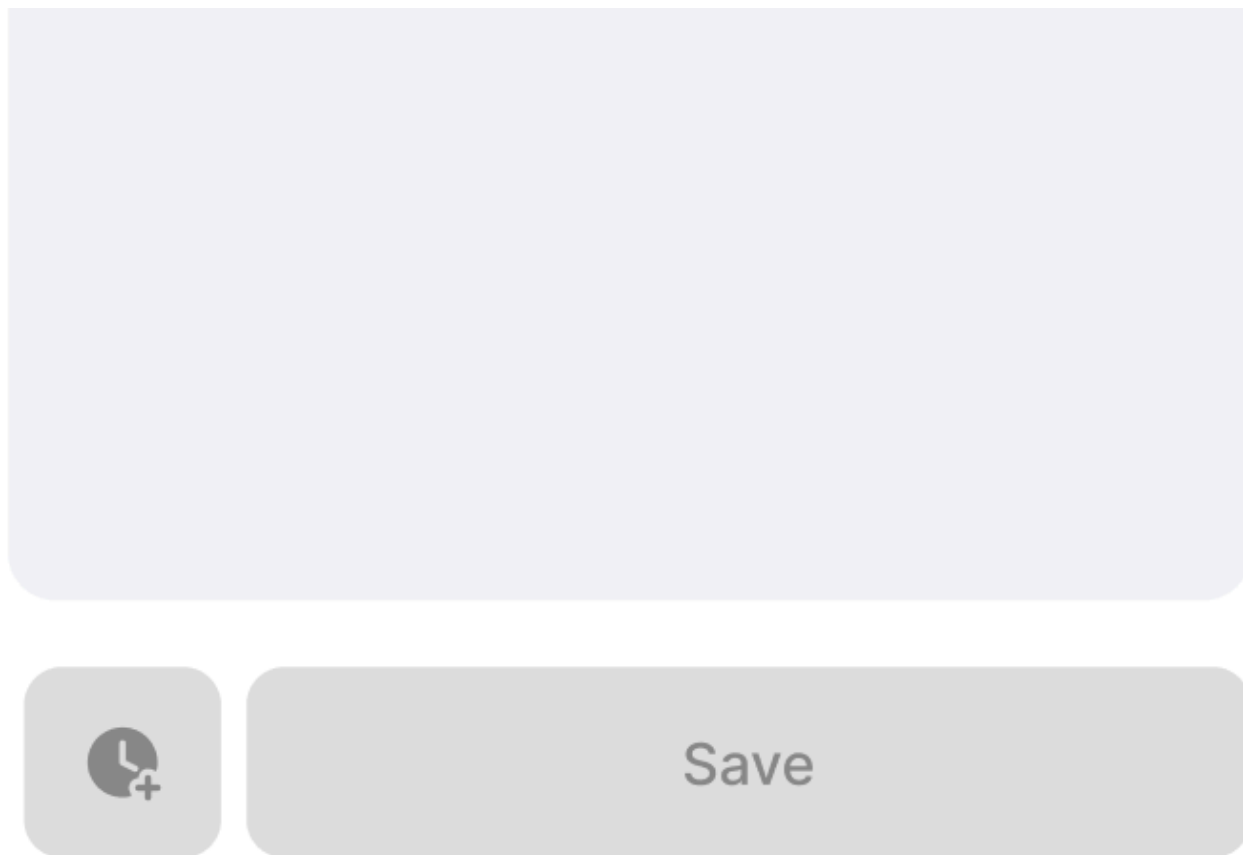
3月1日

02:30

🏷 标签

新建标签





---

## b. 日程详情

在“今日待办”界面，点击日程对应条目，即可跳转到“日程详情”界面。  
在该界面，我们的ai小助手会智能分析今日日程，给出合理的建议。

9:41



# 日程详情



T 学生节策划会议



📍 c楼215



🕒 全天



开始

2月28日

22:00

结束

2月28日

23:00




标签

新建标签



AI建议



根据您今日的日程安排，您的此次安排时间较紧，建议您提前准备会议相关材料并提前出发，以防迟到。

---

## 2. 日程

在“**日程**”界面，您可以通过在日历中点击对应日期，来查看当天的待办及其完成情况

9:41



# Calendar 26 Dec

一	二	三	四	五	六	日
24 廿七	25 廿八	26 廿九	27 三十	28 二月	1 初二	2 初三
3 九九天	4 初五	5 惊蛰	6 初七	7 初八	8 妇女节	9 初十
10 十一	11 十二	12 植树节	13 十四	14 十五	15 消费...	16 十七
17 十八	18 十九	19 二十	20 春分	21 廿二	22 廿三	23 廿四
24 廿五	25 廿六	26 廿七	27 廿八	28 廿九	29 三月	30 初二
31 初三	1 愚人节	2 初五	3 初六	4 清明节	5 初八	6 初九

12天后 1 Dec

☐ 喝8杯水

健康

☐ 移动应用软件开发ddl

作业



☐ 社团活动



今日待办



日程



我的

### 3. 我的

在“我的”界面，您可以进行登录、登出和一些基本的设置操作。

9:41



# Me



## 用户昵称

云同步



修改账户信息

登出账号



今日待办



日程



我的

### 三、项目关键Activity设计与实现

#### a. MainActivity

##### 功能描述：

- 主Activity，作为应用的入口点
- 包含底部导航栏，用于在"今日待办"、"日程"和"我的"三个主要界面之间切换
- 负责加载初始化应用必要的数据和配置

##### 实现方式：

- 使用BottomNavigationView实现底部导航
- 采用Fragment架构，各主要界面作为Fragment进行加载和切换
- 通过ViewPager2与BottomNavigationView联动，实现页面滑动切换
- 使用NavController管理导航逻辑

#### b. TodayTodoFragment (今日待办)

##### 功能描述：

- 显示当日所有待办事项
- 提供快速完成/取消完成待办事项的功能
- 支持添加新待办事项和查看详情

##### 实现方式：

- 使用RecyclerView展示待办事项列表
- 采用CardView设计每个待办事项卡片
- 实现ItemTouchHelper支持左右滑动删除功能
- 使用FloatingActionButton添加新待办事项

## c. AddScheduleActivity (新建日程)

### 功能描述：

- 提供表单用于创建新的待办事项
- 支持设置标题、描述、日期时间、地点和分类标签
- 保存新建的待办事项到本地数据库

### 实现方式：

- 使用MaterialDatePicker和MaterialTimePicker选择日期和时间
- 采用自定义ChipGroup实现分类标签选择
- 使用PlacesAPI或自定义地点选择器实现地点选择
- 通过ViewModel进行数据处理和持久化

## d. ScheduleDetailActivity (日程详情)

### 功能描述：

- 显示待办事项的详细信息
- 提供编辑和删除操作
- 集成AI小助手提供智能建议

### 实现方式：

- 使用ConstraintLayout构建详情页面布局
- 通过Intent接收待办事项ID并加载详细数据
- 实现本地AI模型或调用云端API提供智能建议
- 使用动画效果增强用户体验

## e. CalendarFragment (日程)

### 功能描述：

- 提供月历视图展示各日期的待办事项情况
- 支持选择日期查看特定日期的待办事项
- 显示待办事项完成情况统计

### 实现方式：

- 使用第三方日历控件（如CalendarView或自定义控件）
- 实现日期标记功能，显示有待办事项的日期
- 使用RecyclerView展示选中日期的待办事项列表



- 通过Room数据库查询特定日期的待办事项

## f. ProfileFragment (我的)

### 功能描述：

- 提供用户个人信息管理
- 支持登录、注册和退出登录
- 提供应用设置和数据同步选项

### 实现方式：

- 使用SharedPreferences存储用户偏好设置
- 实现Firebase Authentication或自定义认证系统
- 使用WorkManager实现后台数据同步
- 采用PreferenceFragmentCompat构建设置界面

## g. 数据层实现

### 本地存储：

- 使用Room数据库存储待办事项数据
- 设计Schedule实体类管理待办事项信息
- 实现DAO接口进行数据库操作

### 云端同步：

- 使用Retrofit与后端API交互
- 实现Repository模式管理本地和远程数据源
- 采用WorkManager处理定期同步和冲突解决

## h. 架构设计

### 整体架构：

- 采用MVVM架构模式
- 使用ViewModel管理UI相关数据
- 通过LiveData实现数据驱动的UI更新
- 使用Coroutines处理异步操作
- 采用依赖注入(Hilt/Dagger)简化组件间依赖

### 组件交互：

- Fragment间通过共享ViewModel通信
- Activity跳转通过Intent传递数据
- 使用EventBus或LiveData实现组件间事

## 四、项目主要模块设计

### 1. 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：

- 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。
- 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。
- 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。

### 2. 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盘
  - 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

## 3. 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

## 4. 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

## 五、项目重难点与挑战的分析

### 1. AI集成相关挑战

#### 1.1 AI响应延迟问题

### 问题描述:

AI API请求响应时间不稳定, 在网络条件差或服务负载高时可能出现3-5秒甚至更长的延迟, 导致用户体验下降, 尤其是在任务详情页面加载AI建议时, 界面长时间处于等待状态。

### 解决策略:

#### 1. UI预加载与骨架屏:

- 实现骨架屏(Skeleton UI)显示, 在AI内容加载期间提供视觉反馈
- 使用淡入淡出动画平滑过渡, 降低等待感知

#### 2. 响应超时优化:

- 设置合理的超时阈值(如3秒), 超时后显示友好提示
- 提供手动刷新选项, 让用户控制重试时机

## 1.2 AI内容质量不稳定

### 问题描述:

AI生成内容质量参差不齐, 有时建议过于泛泛而谈, 缺乏针对性; 或者出现内容重复、逻辑不连贯等问题, 影响用户对功能的信任度。

### 解决策略:

#### 1. 提示工程优化:

- 精心设计输入提示(Prompt), 增加任务上下文信息, 多次实验来保证prompt合适
- 针对不同任务类型, 使用专门定制的提示模板

#### 2. 内容过滤机制:

- 设计最小内容标准, 不满足则自动重新生成

#### 3. 用户反馈闭环:

- 提供简单的建议评价机制(有用/无用)
- 根据反馈数据持续优化提示策略

## 2. 数据存储与同步挑战

### 2.1 数据库性能与扩展性

### 问题描述:

随着用户任务量增加, 本地数据库性能可能下降, 尤其是在进行复杂查询(如多条件筛选)或处理大量历史数据时, 导致UI卡顿和响应延迟。

### 解决策略:

#### 1. 数据库优化:

- 创建适当的索引, 优化查询性能
- 实现数据分区, 分别存储活跃和归档任务

#### 2. 查询优化:

- 使用分页加载替代一次性加载全部数据

- 预编译常用查询语句，减少执行时间

### 3. 数据清理策略：

- 提供自动归档功能，移动长期完成的任务
- 实现智能数据保留策略，平衡历史需求和性能

## 2.2 数据安全性与隐私保护

### 问题描述：

用户任务可能包含敏感信息，在数据存储和传输过程中存在隐私泄露风险。

### 解决策略：

#### 1. 隐私设计：

- 采用"隐私设计"原则，最小化数据收集
- 提供数据本地模式，不强制云同步
- 实现数据使用透明度，清晰展示数据流向

#### 2. 用户控制：

- 允许用户选择加密级别和同步范围
- 提供一键数据导出和删除功能
- 实现细粒度权限控制，如"仅标题同步"

## 3. 开发流程与协作挑战

### 3.1 需求变更与迭代管理

### 问题描述：

项目过程中需求可能频繁变更，尤其是AI功能相关需求，由于技术探索性强，可能导致代码重构频繁，进度延迟，开发资源浪费。

### 解决策略：

#### 1. 敏捷开发流程：

- 采用短周期迭代(1-2周/次)，快速验证功能
- 实施MVP策略，优先实现核心价值功能
- 建立需求变更评审流程，评估影响范围

#### 2. 模块化架构：

- 强化接口设计，减少模块间耦合

### 3.2 跨平台兼容性问题

### 问题描述：

不同Android设备和系统版本存在差异，可能导致UI显示异常、功能不一致，增加测试和修复工作量，影响用户体验一致性。

### 解决策略：

#### 1. 兼容性设计原则：

- 遵循Android设计指南和最佳实践

- 使用支持库和AndroidX组件替代原生API

## 2. 自动化测试：

- 使用设备云服务测试多种设备配置
- 实现UI自动化测试验证关键流程

## 3. 用户反馈机制：

- 内置问题反馈渠道，收集设备信息
- 实现远程配置，针对问题设备调整参数
- 建立快速响应机制，及时解决兼容性问题