**SmartProposal Engine MVP设计方案**

**一、系统概述与定位**

**1.1 MVP版本定位**

本MVP版本定位为SmartProposal Engine的核心功能验证版本，聚焦于实现从原始信息输入到商业方案生成的完整链路。系统采用模块化设计，确保与长期目标的兼容性，同时保持当前实现的简洁性。

**1.2 核心价值主张**

* **端到端处理能力**：从原始录音/文档到最终方案的全流程自动化
* **模块化灵活性**：各环节可独立使用，也可串联执行
* **场景适配性**：通过可配置的Prompt模板适应不同业务场景
* **渐进式扩展**：架构设计支持向完整版本平滑演进

**二、系统架构设计**

**2.1 整体架构**

┌─────────────────────────────────────────────────────────┐

│ Web UI Layer (Streamlit) │

├─────────────────────────────────────────────────────────┤

│ Service Layer │

│ ┌───────────┐ ┌───────────┐ ┌──────────────┐ │

│ │ Input │ │ Analysis │ │ Proposal │ │

│ │ Service │ │ Service │ │ Service │ │

│ └───────────┘ └───────────┘ └──────────────┘ │

├─────────────────────────────────────────────────────────┤

│ Core Layer │

│ ┌───────────┐ ┌───────────┐ ┌──────────────┐ │

│ │ Document │ │ Prompt │ │ Model │ │

│ │ Processor │ │ Manager │ │ Interface │ │

│ └───────────┘ └───────────┘ └──────────────┘ │

├─────────────────────────────────────────────────────────┤

│ Storage Layer │

│ Temporary Files / Session Storage │

└─────────────────────────────────────────────────────────┘

**2.2 模块设计原则**

* **高内聚低耦合**：每个服务模块独立完成特定功能
* **统一接口规范**：所有服务遵循相同的输入输出接口
* **配置驱动**：通过配置文件和Prompt模板驱动业务逻辑
* **状态管理**：使用Session State管理跨页面数据流转

**三、功能模块详细设计**

**3.1 输入处理模块（Input Service）**

**功能职责**

* 支持多种输入格式：音频文件（m4a, mp3等）、文档文件（docx, pdf, txt）、Web文本输入
* 统一转换为标准化文本格式
* 保留必要的元数据（文件类型、时长、页数等）

**接口设计**

class InputService:

def process\_input(self,

input\_data: Union[File, str],

input\_type: str,

options: Dict) -> ProcessedInput

**扩展预留**

* 图片OCR接口预留（future: process\_image())
* 视频处理接口预留（future: process\_video())

**3.2 深度分析模块（Analysis Service）**

**功能职责**

* 基于Prompt模板2对转录文本进行商业分析
* 支持多种分析场景（客户访谈、内部会议、商务谈判等）
* 生成结构化的分析报告

**场景模板设计**

ANALYSIS\_TEMPLATES = {

"customer\_interview": "客户访谈深度分析",

"business\_negotiation": "商务谈判要点分析",

"internal\_meeting": "内部会议决策分析",

"requirements\_gathering": "需求收集分析",

"custom": "自定义模板"

}

**3.3 方案生成模块（Proposal Service）**

**功能职责**

* 基于分析结果生成项目建议书
* 支持不同类型的方案模板
* 整合企业能力文档（可选）

**方案模板设计**

PROPOSAL\_TEMPLATES = {

"project\_proposal": "项目建议书",

"quotation\_proposal": "商务报价方案",

"solution\_brief": "解决方案简报",

"meeting\_minutes": "会议纪要及行动计划",

"custom": "自定义模板"

}

**3.4 Prompt管理器（Prompt Manager）**

**功能职责**

* 管理内置Prompt模板
* 支持模板的加载、切换和自定义
* 模板版本控制和热更新

**设计特点**

* 模板与代码分离，便于维护
* 支持模板参数化和动态替换
* 预留模板市场接口

**四、用户界面设计**

**4.1 页面结构**

主页面（Home）

├── 1. 内容输入（Input Processing）

│ ├── 文件上传区

│ ├── 文本输入区

│ ├── 处理选项配置

│ └── 下载转录结果

│

├── 2. 深度分析（Deep Analysis）

│ ├── 上传转录文本

│ ├── 选择分析模板

│ ├── 自定义Prompt（可选）

│ └── 下载分析报告

│

├── 3. 方案生成（Proposal Generation）

│ ├── 上传分析报告

│ ├── 上传能力文档（可选）

│ ├── 选择方案模板

│ └── 下载方案文档

│

└── 4. 一键生成（One-Click Generation）

├── 批量上传所需文件

├── 配置各环节参数

├── 执行全流程处理

└── 批量下载结果

**4.2 交互设计原则**

* **渐进式披露**：基础功能默认展示，高级选项折叠
* **实时反馈**：处理进度可视化，错误信息清晰
* **数据流转**：支持上一步结果自动传递到下一步
* **独立性保证**：每个页面可独立使用

**五、数据流设计**

**5.1 标准数据格式**

# 统一的数据传递格式

class ProcessingResult:

content: str # 主要内容

metadata: Dict # 元数据

source\_type: str # 来源类型

processing\_time: float

model\_used: str

tokens\_consumed: Dict

**5.2 数据流转机制**

1. **Session State管理**：使用Streamlit的session\_state存储中间结果
2. **文件下载/上传**：支持结果导出和导入，确保流程可中断
3. **自动传递**：在一键生成模式下，自动传递各环节数据

**六、配置管理设计**

**6.1 配置文件结构**

[MODELS]

transcription\_model = gemini-2.5-flash

analysis\_model = gemini-2.5-pro

proposal\_model = gemini-2.5-pro

[TEMPLATES]

default\_analysis\_template = customer\_interview

default\_proposal\_template = project\_proposal

[FEATURES]

enable\_custom\_prompts = true

enable\_batch\_processing = true

enable\_capability\_docs = true

**6.2 Prompt模板管理**

prompts/

├── analysis/

│ ├── customer\_interview.md

│ ├── business\_negotiation.md

│ └── internal\_meeting.md

├── proposal/

│ ├── project\_proposal.md

│ ├── quotation\_proposal.md

│ └── solution\_brief.md

└── optimization/

└── transcription\_optimization.md

**七、扩展性设计**

**7.1 向长期目标的演进路径**

**第一阶段（当前MVP）**

* 核心三步流程
* 基础模板支持
* 单文件处理

**第二阶段扩展预留**

* 图片OCR集成（Docling支持）
* 批量文件处理
* 模板市场
* 用户认证系统

**第三阶段扩展预留**

* 工作流引擎集成
* 多模型协同
* 知识库积累
* API开放

**7.2 接口预留设计**

# 预留的扩展接口

class ExtensionInterface:

def register\_processor(self, processor\_type: str, processor: BaseProcessor)

def register\_template(self, template\_type: str, template: BaseTemplate)

def register\_model(self, model\_name: str, model\_config: Dict)

**八、技术实现要点**

**8.1 核心依赖**

* **前端框架**：Streamlit（保持一致性）
* **AI服务**：Google Gemini API
* **文档处理**：python-docx, PyPDF2（基础版）
* **音频处理**：继续使用现有的转录服务

**8.2 代码组织结构**

smart\_proposal\_engine/

├── app.py # Streamlit主应用

├── pages/ # 各功能页面

│ ├── 1\_input\_processing.py

│ ├── 2\_deep\_analysis.py

│ ├── 3\_proposal\_generation.py

│ └── 4\_one\_click\_generation.py

├── services/ # 服务层

│ ├── input\_service.py

│ ├── analysis\_service.py # 基于现有代码

│ ├── proposal\_service.py

│ └── transcribe\_service.py # 复用现有代码

├── core/ # 核心组件

│ ├── prompt\_manager.py

│ ├── model\_interface.py

│ └── document\_processor.py

├── prompts/ # Prompt模板目录

├── config/ # 配置文件

└── utils/ # 工具函数

**8.3 关键设计决策**

1. **保持现有代码复用**：
   * TranscriptionService直接复用
   * DeepAnalysisService作为基础扩展
   * 新增ProposalService遵循相同模式
2. **统一的服务接口**：
3. class BaseService:
4. def process(self, input\_data, template, options) -> ProcessingResult
5. def validate\_input(self, input\_data) -> bool
6. def get\_available\_templates(self) -> List[str]
7. **灵活的模板系统**：
   * 内置模板放在文件系统中，便于更新
   * 支持运行时加载自定义模板
   * 模板支持参数化

**九、MVP实施重点**

**9.1 第一性原理聚焦**

1. **核心功能优先**：只实现三步主流程
2. **简化配置**：使用默认配置，减少用户选择
3. **快速验证**：优先实现端到端流程，后续优化细节

**9.2 不在MVP范围内的功能**

* 用户认证和权限管理
* 数据持久化存储
* 高级错误处理和重试机制
* 性能优化和缓存
* 多语言支持

**十、总结**

本设计方案基于现有音频转录系统，通过模块化和可扩展的架构设计，实现了SmartProposal Engine的MVP版本。方案充分考虑了与长期目标的兼容性，同时保持了当前实现的简洁性。通过统一的服务接口、灵活的模板系统和清晰的数据流设计，确保系统既能满足当前需求，又能支持未来的渐进式扩展。