|  |  |
| --- | --- |
|  | |
| **《个人知识库助手》**  **项目概要设计**  **V1.0** | |
|  |

|  |
| --- |
|  |

**版 本 历 史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 日期 | 备注 |
| 1.0.0 |  |  | 2025-08-27 | 创建 |

**目 录**

[第一部分 引言 5](#_Toc1719706813)

[一、编写目的 5](#_Toc566643133)

[二、读者对象 5](#_Toc607892380)

[三、术语与缩写解释 5](#_Toc552879502)

[1、 知识库（Knowledge Base） 5](#_Toc1223225396)

[2、 RAG（Retrieval-Augmented Generation，检索增强生成） 5](#_Toc16119017)

[3、 LLM（Large Language Model，大语言模型） 6](#_Toc1772196933)

[4、 Embedding（向量化表示） 6](#_Toc1671454255)

[5、 Vector Database 6](#_Toc1593561293)

[6、 Faiss（Facebook AI Similarity Search） 6](#_Toc2129384684)

[7、 Retrieval（检索） 7](#_Toc549990903)

[8、 Similarity Search（相似度搜索） 7](#_Toc869882021)

[9、 Chunk（分块） 7](#_Toc1151635434)

[10、 TOP-K 7](#_Toc1754038344)

[11、 Context（上下文） 7](#_Toc332386050)

[12、 Prompt 7](#_Toc1427385111)

[13、 DRF(Django REST Framework) 7](#_Toc1072976605)

[14、 API（应用程序接口） 8](#_Toc142002252)

[15、 SSE（Server-Sent Events，服务端推送事件） 8](#_Toc1920491273)

[四、参考资料 8](#_Toc811397690)

[第二部分 项目概述 9](#_Toc1787173308)

[一、项目描述 9](#_Toc902269086)

[二、项目功能描述 10](#_Toc749482712)

[1. 文件处理 10](#_Toc1543481455)

[2. 知识库 11](#_Toc1119807537)

[3. 智能问答 11](#_Toc1845969258)

[4. 多人协作 11](#_Toc608415990)

[5. 用户交互与结果回传 12](#_Toc405787463)

[第三部分 设计约束 12](#_Toc2081895054)

[一、需求约束 12](#_Toc1291548759)

[1.应遵循的技术标准 12](#_Toc547978168)

[2.接口/协议标准 12](#_Toc1654118219)

[3.用户界面标准 13](#_Toc1858191892)

[4.软件质量 13](#_Toc1155870548)

[5.网络体系结构 13](#_Toc59514073)

[二、隐含约束 14](#_Toc933933640)

[第四部分 个人知识库助手方案设计 15](#_Toc1171989565)

[一、安全设计 15](#_Toc1831711006)

[二、 个人知识库助手业务流程 15](#_Toc457904247)

[三、业务功能概要结构 16](#_Toc618067211)

[1、知识库导入删除模块 16](#_Toc1813612042)

[1.1、 导入知识库 16](#_Toc1007895150)

[1.2、 删除知识库内容 17](#_Toc1487949232)

[2、检索与回答模块 17](#_Toc817763829)

[2.1、关键词模糊检索 17](#_Toc614449847)

[2.2、标签限定检索 18](#_Toc1820335282)

[2.3、智能上下文加入 18](#_Toc97665292)

[四、 模块定义 18](#_Toc1687426452)

[第五部分 E-R实体设计 19](#_Toc1962337534)

[一、E-R实体结构图 20](#_Toc2018156565)

[二、实体描述 20](#_Toc351340494)

[1、CustomUser实体描述 20](#_Toc1602027194)

[2、UserProfile 实体描述 20](#_Toc772942003)

[3、Knowledge Base实体描述 21](#_Toc1100823207)

[4、Document实体描述 21](#_Toc998025002)

[5、AIAgent实体描述 21](#_Toc1892749540)

[6、AgentSession实体描述 22](#_Toc799308817)

[7、ChatSession实体描述 22](#_Toc1606440992)

[8、ChatMessage实体描述 22](#_Toc151053356)

[9、Integration实体描述 23](#_Toc733720223)

[10、DocumentViewSet实体描述 23](#_Toc750506103)

[第六部分 总体设计 24](#_Toc699031524)

[一、个人知识库助手逻辑架构设计 24](#_Toc240354794)

[1、创新的向量型数据库与关系型数据库结合架构 24](#_Toc461214347)

[2、可解释性优先目标 24](#_Toc1854902072)

[3、高度个性化的用户界面 24](#_Toc299868868)

[二.物理架构设计 25](#_Toc1395147987)

[三.技术架构设计 25](#_Toc879407990)

[第七部分 用户界面设计 28](#_Toc2131579874)

[一、桌面布局设计 28](#_Toc1853052234)

[1、登录界面风格 28](#_Toc1497475201)

[2、主桌面风格 28](#_Toc1797708269)

[二、业务界面风格展示 29](#_Toc713463736)

[1、风格展示 29](#_Toc837940785)

[第八部分 运行环境和部署 30](#_Toc467988450)

[一、运行环境 30](#_Toc1327913583)

[1、客户机器环境 30](#_Toc510792419)

[2、开发环境要求 30](#_Toc565653742)

[二、系统性能要求 30](#_Toc867856387)

第一部分 引言

### 一、编写目的

编写本文的主要目的是把需求分析得到的用例模型转换为软件结构和数据结构。设计软件结构的具体任务是：将一个复杂系统按功能进行模块划分、建立模块的层次结构及调用关系、确定模块间的接口及人机界面等。数据结构设计包括数据特征的描述、确定数据的结构特性、以及数据库的设计。

本设计是指导详细设计和项目实施的重要指导性文件，也是进行系统集成测试和重要依据。

### 二、读者对象

该文档的读者为用户代表、软件分析人员、开发管理人员和测试人员。

### 三、术语与缩写解释

#### 1、 **知识库（Knowledge Base）**

知识库是存储、组织和管理知识的系统化集合，可以包含结构化（数据库表）、半结构化（JSON、Markdown）、非结构化（Word、PDF、图片）等多种形式的数据。  
在本项目中：

1. 用户上传的文档（PDF、Word、txt 等）会被解析、切分成较小的内容片段（Chunk）。
2. 每个片段会通过 **Embedding 模型** 转换为向量。
3. 文本与向量共同存放在 **向量数据库** 中。  
    作用：支撑后续 **检索（Retrieval）+ 问答（Q&A）** 过程，实现知识增强型智能助手。

#### **RAG（Retrieval-Augmented Generation，检索增强生成）**

RAG 是一种结合 **信息检索系统** 与 **大语言模型（LLM）** 的方法：

1. 接收用户问题 → 通过向量检索在知识库中找到相关内容。
2. 将检索结果作为上下文拼接到 **Prompt** 中。
3. 输入 LLM 生成回答。  
   优点：

* 降低幻觉（Hallucination）问题。
* 确保答案更贴近用户知识库内容。
* 支持动态扩展：只需更新知识库即可增强模型知识。

#### **LLM（Large Language Model，大语言模型**）

LLM 是利用深度学习（特别是 Transformer 架构）训练的大规模神经网络，具备自然语言理解与生成能力。该项目中主要功能如下：

1. LLM负责生成最终答案。
2. 输入 = 用户问题 + 检索到的上下文
3. 输出 = 针对问题的回答

常见模型：ChatGPT、LLaMA、Mistral、DeepSeek

#### Embedding（向量化表示）

Embedding 是把文本（词、句子、文档）转化为**高维稠密向量**的过程，使语义相近的文本在向量空间中距离更近。通常由预训练模型（OpenAI Embedding API、 Sentence-BETR）完成，支持高效相似度计算与检索，常用于FAQ匹配、语义搜索和知识库检索。

#### Vector Database

向量数据库，专门存储与检索高维向量的数据库，支持近似最近邻（ANN）搜索。常见的有Faiss、Chroma等。

#### Faiss（Facebook AI Similarity Search）

Meta 开源的高性能相似度搜索库，适合处理百万/亿级向量集合，支持 CPU/GPU。内置多种索引方式，如

#### Retrieval（检索）

从知识库或向量数据库中找到与用户查询最相关的文档片段，作为上下文传递给大模型。通过输入用户问题Query，将Query转化为向量，再进行相似度搜素，最后输出Top-K的相关文档片段。

#### Similarity Search（相似度搜索）

相似度搜索，在向量空间中，根据余弦相似度或欧氏距离等指标，查找与输入向量最接近的 K 个向量。

#### Chunk（分块）

将长文档切分为较小的片段（例如 200-800 tokens），保证检索粒度合适并避免超过 LLM 输入限制。

#### TOP-K

检索时返回相似度排名前K的文档片段，K的选择会影响结果质量和生成速度。

#### Context（上下文）

检索出的文档片段，作为 LLM 生成答案时的“参考资料”，也叫提示上下文。

#### Prompt

提示词，给 LLM 的输入内容，包含用户问题、系统指令和检索到的上下文，决定生成结果的风格与准确性。

#### DRF(Django REST Framework)

Django 框架的 REST API 扩展，用于后端接口开发，可以上传文档、搜索知识库、对话问答等

#### API（应用程序接口）

应用程序接口（Application Programming Interface），项目中通过 API 实现前后端交互，例如上传文档、搜索、对话等。

#### SSE（Server-Sent Events，服务端推送事件）

一种实时单向推送技术，常用于将 LLM 的流式回答传递给前端。基于HTTP协议，轻量，适合文本传输，带来更好的用户体验。

### 四、参考资料

1. LangChain Documentation
2. Django REST Framework Documentation
3. OpenAI API Documentation
4. Faiss Documentation
5. RAG技术综述

第二部分 项目概述

### 一、项目描述

本项目旨在构建个人知识库助手，通过结合知识库管理、向量数据库检索与大语言模型（LLM）生成，实现智能化的文档问答与知识查询。用户可以上传多种格式的资料（如 PDF、Word、Markdown 等），系统会自动完成文档解析、分块（Chunk）、向量化（Embedding）及存储，从而形成可检索的个人知识库。

当用户提出问题时，系统会依托检索增强生成（RAG）流程，先从知识库中检索最相关的内容片段，再将其与用户问题一同传递给 LLM，生成更加准确、可信、可追溯的回答。

在架构设计上，系统后端采用 Django + Django REST Framework（DRF） 实现 API 服务，负责文档处理、知识库维护与对外接口；前端基于 React 实现，提供便捷的文档上传与问答交互界面；向量数据库采用 Faiss 等高性能检索工具；大语言模型调用则支持 DeepSeek、OpenAI 或本地模型，确保系统具备灵活的扩展性和适配性。



### 二、项目功能描述

文档处理包含四个主要业务功能：文件上传解析、文档分块与向量化、向量数据库入库、知识库维护管理。  
 智能问答包含九个主要业务功能：自然语言检索、相似度搜索、上下文选择、RAG 问答生成、答案流式返回、引用标注、历史对话查询、答案反馈与修正、多用户协作与权限控制。

#### 文件处理

* 1. **文件上传解析**。用户可通过系统上传 PDF、Word、TXT 等常见格式文件。系统对上传文件进行完整性和合法性校验，并调用解析模块将其内容抽取为结构化文本，统一转为 Markdown 格式，以保持段落、标题、表格等排版信息。该过程确保不同来源的文档能够标准化存储和处理。
  2. **文档分块与向量化。**为适配大语言模型的输入限制，系统会自动将长文档切分为多个较小片段（Chunk）。随后，利用 Embedding 模型将每个片段转化为高维向量，向量与原文档片段保持一一对应关系。该功能实现了对知识的细粒度表达，为后续的相似度检索奠定基础。
  3. **向量数据库入库。**系统将生成的片段向量存入 Faiss 向量数据库，同时在 MySQL 数据库中保存对应的 Markdown 文件与元数据（如文件来源、页码、段落位置）。此功能保证了“语义向量 ↔ 原文片段”的映射关系，确保在检索结果返回时能准确追溯到原始文档内容。

#### 知识库

* 1. **知识库维护管理。**系统提供知识库的维护与管理能力，包括文档版本更新、索引重建、冗余数据清理以及异常处理。当文件重新上传或模型版本更新时，可自动触发索引重建，保证知识库中的数据长期一致、可靠和可扩展。
  2. **RAG问答生成。**系统采用检索增强生成（RAG）模式，将用户问题与选定的上下文片段共同传递给大语言模型，生成更准确、基于事实的回答。该功能有效减少了模型的“幻觉”问题，使回答更贴近知识库内容。

#### 智能问答

* 1. **自然语言索引。**用户通过自然语言输入问题，系统将问题文本转化为向量，并在向量数据 库中执行相似度检索，快速定位与用户问题语义最接近的文档片段。该功能实现了基于语义的智能搜索，超越了传统关键词匹配。
  2. **相似度检索。**在检索过程中，系统会利用余弦相似度、欧氏距离等度量方法，选取与用户问题最相关的前 Top-K 片段。通过精确的相似度计算，该功能保证检索结果与用户需求高度相关。
  3. **上下文选择。**检索完成后，系统将返回的候选片段展示给用户。用户可自动选择系统推荐的结果，或手动挑选最合适的片段作为上下文，从而增强回答的针对性与可信度。
  4. **引用标注。**在生成的回答中，系统会自动添加引用信息，标注对应的文档片段来源。用户可以通过引用快速定位原始内容，确保回答的可追溯性和透明度。
  5. **历史对话查询**。系统会保存用户的提问、检索上下文及最终回答，形成可回溯的知识交互记录。用户可以在历史记录中查找先前的问题与答案，便于知识的积累和复用。

#### 用户交互与结果回传

* 1. **答案流式返回。**为提升交互体验，系统通过 Server-Sent Events（SSE）技术将大模型的回答以流式形式逐步推送至前端。用户可实时查看回答生成过程，获得与 DeepSeek 类似的交互体验。
  2. **答案反馈与修正。**用户可对回答结果进行反馈，例如标注“正确/不准确”，或提交补充信息。系统将收集这些反馈，并用于改进知识库内容、检索策略或提示词设计，形成持续优化闭环。

第三部分 设计约束

### 一、需求约束

#### 1.应遵循的技术标准

1. 编码与命名规范
   1. 后端遵循《PEP 8》与《Django 编码规范》，REST 资源命名用复数名词、下划线风格。
   2. 前端遵循 Airbnb JavaScript/React Style Guide；组件采用 PascalCase，文件/路径采用小写短横线。
   3. 接口采用 RESTful 语义与一致的错误码（HTTP + 业务码），时间统一使用 ISO-8601（UTC 存储、前端本地化展示）。
2. 数据与文档规范。
   1. 接口文档采用 OpenAPI 3.0（Swagger）自动生成与校验。
   2. 配置采用 12-Factor 原则：.env 环境变量 + YAML/JSON 配置。
   3. 日志采用 JSON 行日志（request\_id、user\_id、latency、status）。

#### 2.接口/协议标准

1. 对外API
   1. 协议：HTTPS + JSON；统一前缀 /api/v1，向后兼容的版本策略。
   2. 鉴权：OAuth2 / JWT（短期 token + 可选 refresh），支持 RBAC。
   3. 分页、排序、过滤统一；接口幂等要求 PUT/DELETE 幂等。
   4. 速率限制与配额控制（全局与用户级）。
2. 对内集成
   1. 向量检索与 LLM 通过 LangChain 标准接口，模型与嵌入提供可插拔适配层（OpenAI/DeepSeek/本地）
   2. 回调/异步：支持 Webhook；任务编排用 Celery + Redis（或 RQ）处理解析/向量化/索引重构等耗时任务。

#### 3.用户界面标准

1. UI 组件：React + Ant Design（或自定义 + Tailwind），支持深色/浅色模式与响应式布局。
2. 交互规范：统一按钮/表单/消息反馈模式，关键操作二次确认。
3. 文本检索与回答界面遵循一致搜索框与结果卡片样式，支持来源可追溯（cite 原文片段、高亮命中）。

#### 4.软件质量

1. 正确性
   1. RAG 流程：解析→分块→向量化→检索→重排→生成，流程可观测且可回放。
   2. 单元测试覆盖率≥80%，关键流程集成测试与契约测试齐备。
2. 健壮性
   1. 目标可用性：≥99.5%；后端与向量库/对象存储具备健康检查与降级策略（如回答回退为“仅检索摘要”）。
   2. 支持 7×24 稳定运行，异常自动告警。
3. 易用性
   1. 界面图形化、表单化；支持拖拽上传、批量导入、进度与错误可视化。
   2. 提供“来源定位”一键跳转与导出引用。
4. 安全性
   1. 存储：敏感字段（API Key、用户口令、访问令牌）加密（AES-256），散列采用 bcrypt/argon2。
   2. 权限：最小权限原则。
   3. 审计：关键操作审计日志与溯源。
5. 可扩展性
   1. 向量库、嵌入模型、LLM 以适配器模式可热插拔。
   2. 支持多用户（可选逻辑隔离/物理隔离）。

#### 5.网络体系结构

1. 客户端：浏览器端 React SPA，支持公司/校园网与公网访问。
2. API网关/反向代理：Nginx/Traefik 统一入口，TLS 终止、限流与路由。
3. 应用层：Django + DRF 提供 REST API；异步任务队列（Celery + Redis）。
4. 数据与存储：MySQL（元数据/权限/审计）、FAISS（向量检索）、MinIO/S3（原文与切片存储）、Redis（缓存/队列）
5. 模型与外部服务：LLM/Embedding 提供商（DeepSeek/OpenAI/本地推理服务）通过公网安全访问；可选本地推理网关。

### 二、隐含约束

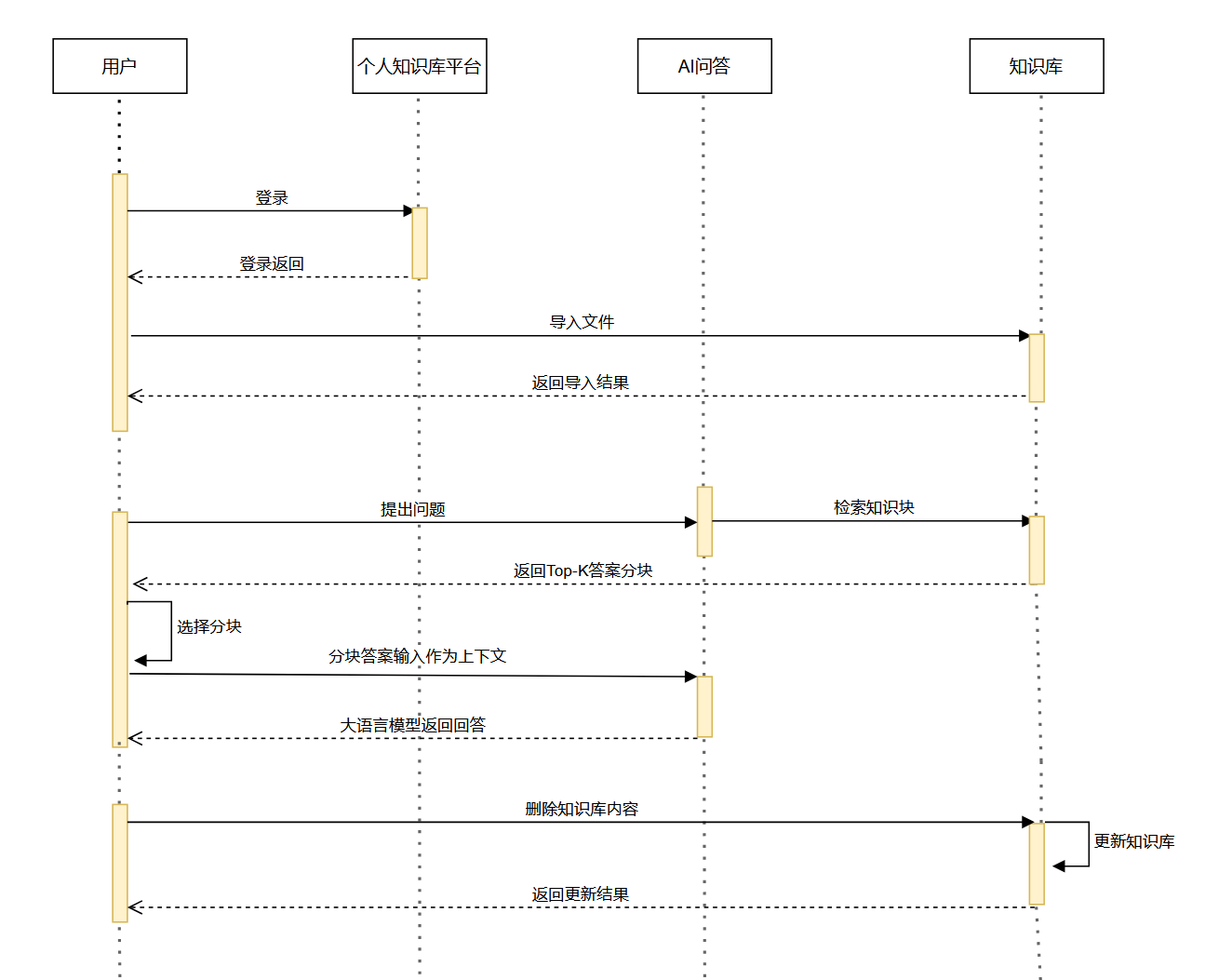
1. 用户具备基本计算机操作能力；经简短培训可独立完成上传、检索与溯源操作。
2. 系统可在主流现代浏览器运行（Chrome/Edge/Firefox/Safari 最新两个大版本）；移动端提供基础适配。
3. 可变参数全部外置化（环境变量/配置中心/数据库参数表），无需重启即可生效（若安全策略允许）。
4. 导入解析器与分块策略可配置（页长、重叠、规则模板）；停用词/同义词词典可维护。
5. 合规要求导致的“删除权/导出权”须在产品层面提供一键操作与后台审计。

第四部分 个人知识库助手方案设计

### 一、安全设计

个人知识库助手通过“用户账号 + 验证码”为KEY在接入层完成绑定与校验管理。  
 系统复用统一用户数据库，用户名/密码在后端校验，并按角色自动控制文档访问权限。

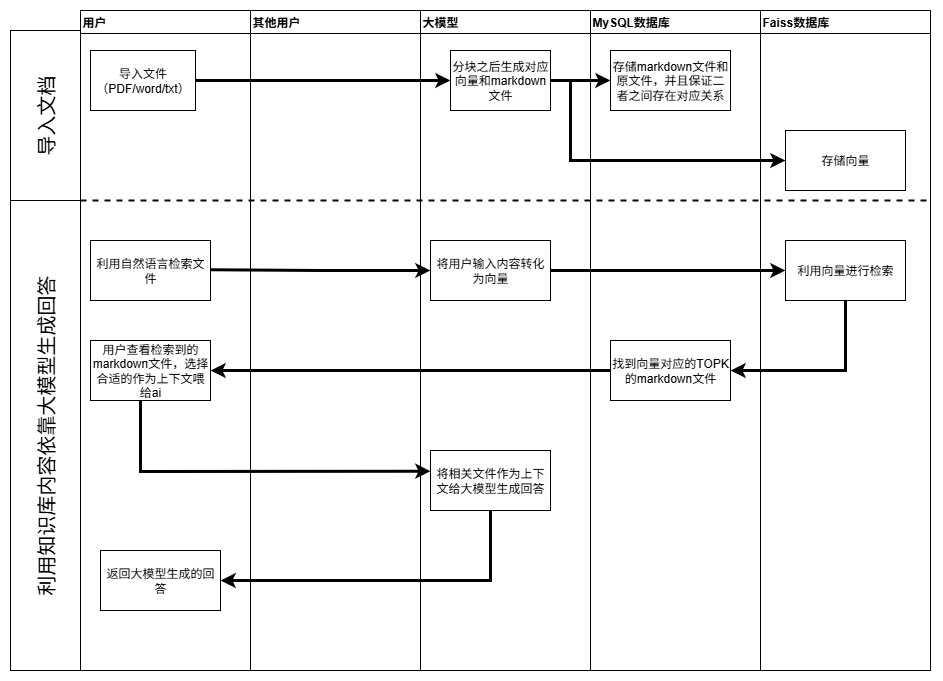
### 个人知识库助手业务流程



**说明：**

1. 用户输入账号密码登录
2. 平台返回登录结果信息
3. 用户导入文件到知识库
4. 返回导入是否成功
5. 用户在AI问答单元中提问
6. 问答单元在知识库中检索匹配的分块
7. 返回Top-K分块给用户
8. 用户选择分块作为上下文给问答单元
9. 大模型返回结果
10. 用户选择性删除知识库内容
11. 知识库更新返回更新信息

### 三、业务功能概要结构



#### 1、知识库导入删除模块

##### 导入知识库

1. 用户上传文件（支持 PDF、Word、Markdown、TXT 等），系统进行大小与格式校验。
2. 用户选择目标知识库及分块，系统校验写入权限。
3. 系统解析文件内容，去除无效信息（页眉、页脚、重复内容），并按规则切分为知识块。
4. 系统调用 Embedding 模型对知识块向量化，写入向量数据库，并生成关键词倒排索引。
5. 系统展示预计导入条目数与配额占用情况，用户确认后执行导入。
6. 导入过程中支持进度展示与失败重试，完成后返回结果统计并生成日志。

##### 删除知识库内容

1. 用户选择删除范围（整库、分块、文档或片段），支持标签或时间筛选。
2. 系统校验用户删除权限，并提供软删除（可恢复）与硬删除（不可恢复）两种方式。
3. 删除时同步清理对应的向量与关键词索引，并更新知识库目录与统计。
4. 删除完成后系统返回操作结果，支持操作日志与回收站恢复。

#### 2、检索与回答模块

##### 2.1、关键词模糊检索

1、通过自然语言给出检索需求。

2、系统根据设定的维度与模型对自然语言进行向量化处理。

3、将向量与向量数据库faiss内的向量进行向量相似度搜索，获得最接近的的K个向量。

4、后端根据检索到的向量从mysql数据库得到原文内容与位置。

5、显示K个检索到的原文内容与来源位置。

6、点击搜索结果可自动跳转到原文档，回归原有上下文。

##### 2.2、标签限定检索

1、用户可以对每一个文档自定义标签或由AI生成标签。

2、用户可以提前选择标签限定查询范围。

3、后端只会在有此标签的向量范围内进行比较并返回结果。

##### 2.3、智能上下文加入

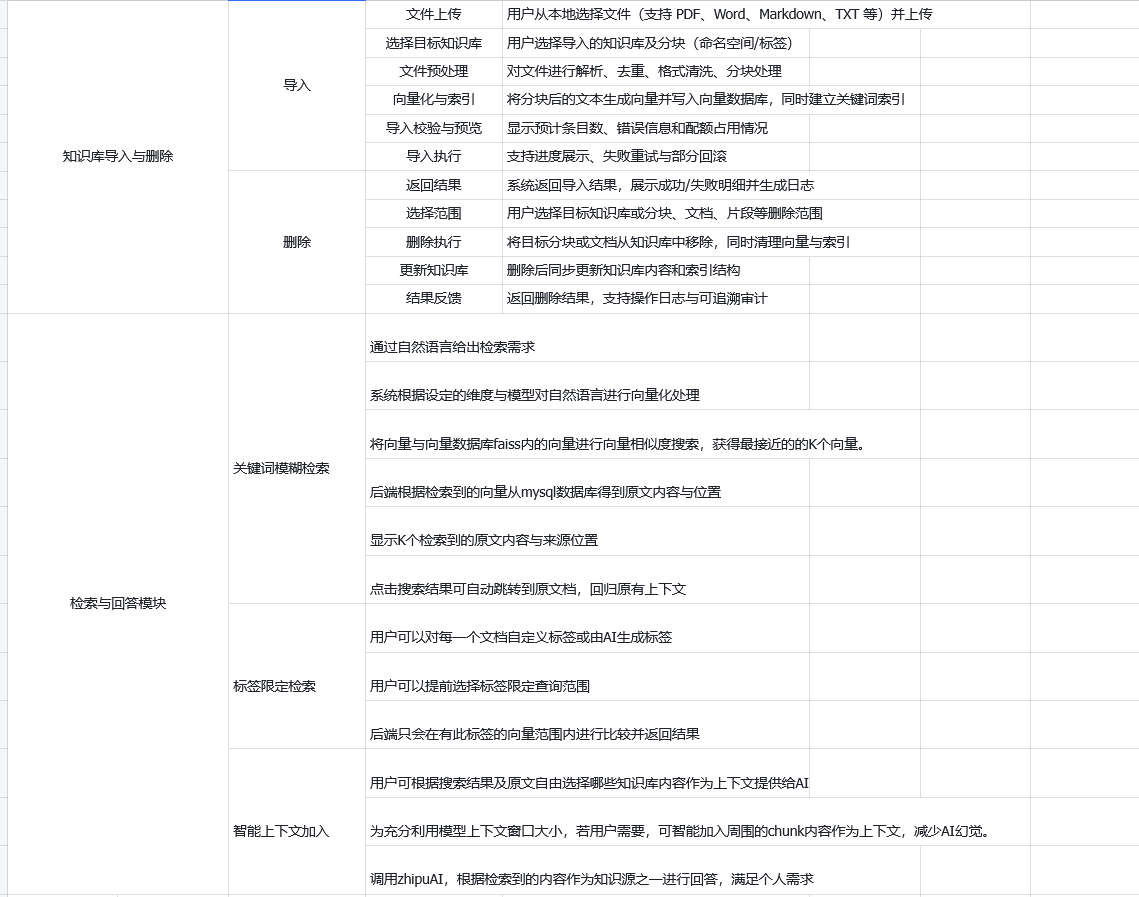
1、 用户可根据搜索结果及原文自由选择哪些知识库内容作为上下文提供给AI。

2、为充分利用模型上下文窗口大小，若用户需要，可智能加入周围的chunk内容作为上下文，减少AI幻觉。

3、调用zhipuAI，根据检索到的内容作为知识源之一进行回答，满足个人需求。

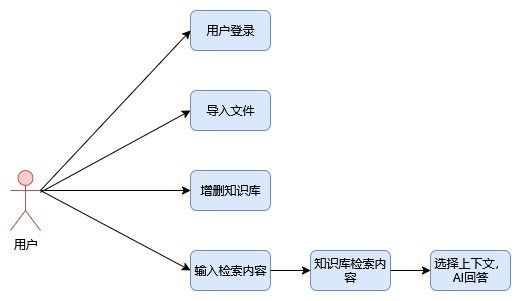
### 模块定义

目前分别设计了知识库导入与删除和检索与回答两个模块。分别有以下组件和具体功能。



### 五：用例分析

#### 1.1、 用户用例



##### 1.1.1 用户登录用例描述

|  |  |
| --- | --- |
| ID | A001 |
| 用例名称 | 用户登录 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 打开登录页面 2. 输入用户名及密码 3. 输入验证码 4. 点击登录按钮 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

##### 1.1.2 用户注册用例描述

|  |  |
| --- | --- |
| ID | A001 |
| 用例名称 | 用户注册 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 打开注册页面 2. 输入用户名及密码还有邮箱 3. 点击注册按钮 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

##### 1.1.3 用户注销用例描述

|  |  |
| --- | --- |
| ID | A001 |
| 用例名称 | 用户注销 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 打开注销页面 2. 输入用户名及密码 3. 点击注销按钮 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

#### 1.2、文件增删用例

##### 1.2.1 导入文件用例说明

|  |  |
| --- | --- |
| ID | A002 |
| 用例名称 | 导入文件 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 选择导入的知识库或创建 2. 选择本地文件导入知识库 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

##### 1.2.1 删除文件用例说明

|  |  |
| --- | --- |
| ID | A002 |
| 用例名称 | 删除文件 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 选择删除文件所在的知识库 2. 选择要删除的文件 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

#### 1.3、增删知识库用例说明

|  |  |
| --- | --- |
| ID | A003 |
| 用例名称 | 增删知识库 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 进入目标知识库 2. 选择目标的分段内容 3. 增加、删除该内容 4. 更新知识库 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

#### 1.4、输入检索内容

|  |  |
| --- | --- |
| ID | A004 |
| 用例名称 | 输入检索内容 |
| 父用例ID | - |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 用户输入想要检索的内容（ 2. 大模型转化为文本向量 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

#### 1.5、知识库检索内容用例说明

用户可以选择检索内容所用的大模型，目前有智普AI、deepseek、通意千问可以作为选择

|  |  |
| --- | --- |
| ID | A005 |
| 用例名称 | 知识库检索内容 |
| 父用例ID | A004 |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 选择大模型 2. 大模型计算文本向量相似度 3. 根据topk原则检索知识库 4. 返回k个分段内容 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

#### 1.6、选择上下文AI回答用例说明

用户可以选择生成回答的大模型，目前有智普AI、deepseek、通意千问可以作为选择

|  |  |
| --- | --- |
| ID | A006 |
| 用例名称 | 选择上下文AI回答 |
| 父用例ID | A005 |
| 主要执行者 | 主用户 |
| 前置条件 | - |
| 事件流 | 1. 用户选择大模型 2. 用户选择分段内容作为上下文 3. AI大模型生成回答 |
| 可选事件流 | - |
| 异常事件流 | - |
| 后置条件 | - |

第五部分 E-R实体设计

### 一、E-R实体结构图

根据Django的框架模式，我设计了以下几个APP模块：

account模块：

功能：负责用户管理、认证和授权，包含用户注册、登录和个人资料管理。

主要数据模型：

·User：用户基础信息模型

Profile：用户扩展信息模型，与User通过OneToOneField关联

account/

├── models/

│ ├── user\_models.py # 用户模型

│ └── profile\_models.py # 用户资料模型

├── serializers.py # 数据序列化器

├── views.py # API视图

└── urls.py # URL路由配置

Knowledge模块：

功能：负责知识库的创建、文档上传、解析、分块和向量化，为AI提供知识来源。

主要数据模型：

KnowledgeBase：知识库模型

DocumentModels：文档模型

ChunkModels：文档分块模型

MarkdownModels：Markdown格式内容模型

knowledge/

├── models/

│ ├── knowledge\_base\_models.py # 知识库模型

│ ├── document\_models.py # 文档模型

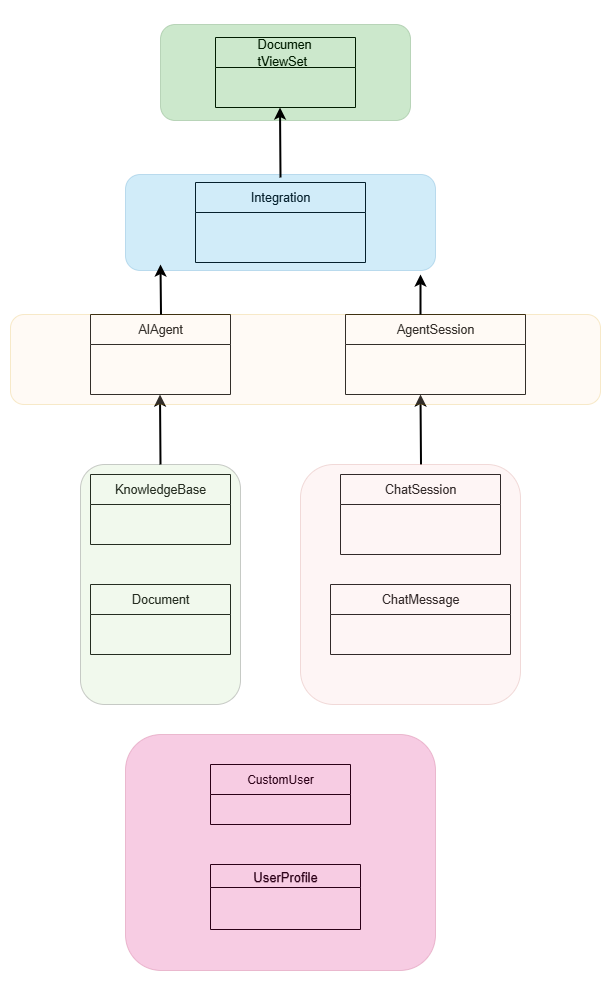
│ ├── chunk\_models.py # 分块模型

│ └── markdown\_models.py # Markdown模型

├── serializers.py # 数据序列化器

├── views.py # API视图

└── urls.py # URL路由配置



### 二、实体描述

#### 1、CustomUser实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 用户ID（PK） | INT |
| 2 | username | 用户名 | Varchar |
| 3 | email | 邮箱 | Varchar |
| 4 | password | 密码 | Varcahr |

#### 2、UserProfile 实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | profileid | 资料ID | Int |
| 2 | userid | 用户ID | Int（FK） |
| 3 | realname | 姓名 | Varchar |
| 4 | avatar | 头像 | Varchar |
| 5 | bio | 简介 | Text |

#### 3、Knowledge Base实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 知识库ID（PK） | INT |
| 2 | owner\_uer\_id | 所属用户ID（FK） | INT |
| 3 | name | 名称 | VARCHAR(120) |
| 4 | description | 描述 | Text |
| 5 | embed\_model | 向量模型标识 | VARCHAR(80) |
| 6 | created\_at | 创建时间 | DATETIME |

#### 4、Document实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 文档ID（PK） | INT |
| 2 | kb\_id | 知识库ID（FK） | INT |
| 3 | title | 标题 | VARCHAR(200) |
| 4 | file\_type | 文件类型 | VARCHAR(40) |
| 5 | storage\_uri | 存储地址 | VARCHAR(255) |
| 6 | chunk\_count | 分块数量 | INT |
| 7 | created\_at | 创建时间 | DATETIME |

#### 5、markdown实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 代理ID(PK) | String |
| 2 | creater\_id | 所属用户(FK) | String |
| 3 | Knowledge\_base\_id | 关联知识库(FK) | String |
| 4 | title | 名称 | VARCHAR(120) |
| 5 | content | 内容 | VARCHAR(300) |
| 6 | vector\_id | 向量数据库行id | String |
| 7 | number | 是所属document的第几个markdown | int |
| 8 | create\_at | 创建时间 | datetime |

#### 7、ChatSession实体描述

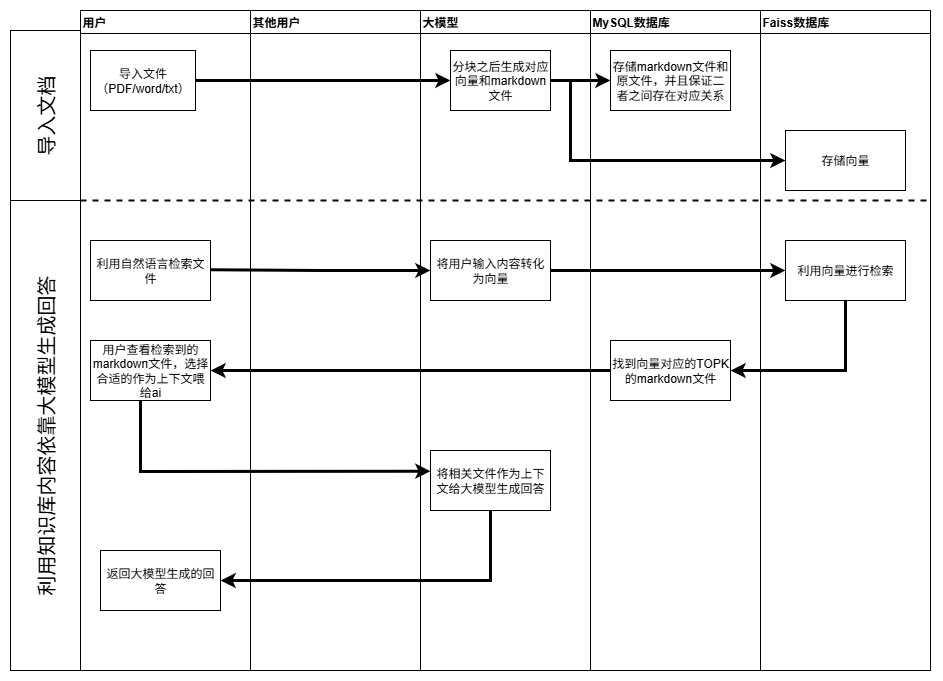
|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 对话会话ID(PK) | INT |
| 2 | agent\_session\_id | 代理会话ID(FK) | INT |
| 3 | topic | 主题 | VARCHAR(200) |
| 4 | created\_at | 创建时间 | DATETIME |

#### 8、ChatMessage实体描述

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **英文名** | **中文名** | **数据类型** |
| 1 | id | 消息ID(PK) | INT |
| 2 | char\_session\_id | 对话会话ID(FK) | INT |
| 3 | role | 角色 | VARCHAR(20) |
| 4 | content | 内容 | TEXT |
| 5 | created\_at | 发送时间 | DATETIME |

第六部分 总体设计

### 一、个人知识库助手逻辑架构设计



#### 1、创新的向量型数据库与关系型数据库结合架构

结合了向量数据库的语义搜索能力和关系型数据库的结构化数据管理优势，实现跨模态检索，用户可以使用自然语言进行模糊搜索与精确搜索，系统可自动发现多篇文档之间潜在的语义关联，便于整理、检索个人知识 。

#### 2、可解释性优先目标

将用户的搜索内容向量化后，根据个人需求获得前k个相关答案并展现原文与原文位置，点击可达原文整体，还原原文信息。降低过程中的AI幻觉，增强可解释性。

#### 3、高度个性化的用户界面

使用react的chat-ui库，提供高度定制化的ui界面，满足用户的个性化需求，模块化组件，丰富用户选择，提供差异化服务。

### 二.物理架构设计

**1）服务端**  
系统服务端可部署在云平台或本地服务器上，由系统管理员通过浏览器（IE/Chrome 等）进入后台管理界面，实现文档管理、知识库维护、用户权限及系统配置等业务管理操作。

**2）具体应用**  
服务端作为知识库数据与检索的提供者，集成了文档解析、文本切分、向量生成、索引管理等功能模块，并提供 API 接口。移动端、PC 端等终端用户均可通过服务端获取知识检索和问答服务。

**3）客户端**  
系统采用 B/S 结构模式，PC 客户端只需具备浏览器（IE/Chrome/Edge）即可访问，不需要安装额外软件。用户通过浏览器上传文档、发起搜索、查看答案与溯源信息。

。

### 技术架构设计

1. 系统概述

Knowledge Assistant是一个基于Django框架构建的知识库管理与智能交互平台，支持用户创建知识库、上传文档、进行AI对话等功能

2. Django架构特点

2.1 MVT架构模式

本项目严格遵循Django特有的MVT（Model-View-Template）架构模式，而非传统的MVC架构。在Django中，没有独立的Controller层，这是Django框架的核心特点之一：

Model层：负责数据定义和数据库交互

View层：处理业务逻辑和请求响应（包含了传统MVC中Controller的功能）

Template层：负责页面渲染（本项目主要是API，此层作用较弱）

2.2 项目目录结构

d:\knowledge\_assistant\knowledge\_assistant\

├── account/ # 用户账户应用

├── chat/ # 聊天功能应用

├── knowledge/ # 知识库管理应用

├── knowledge\_assistant/ # 项目主配置目录

├── manage.py # Django管理脚本

└── media/ # 媒体文件存储目录

3. Django MVT各层详解

3.1 Model层设计

Model层是Django应用的基础，负责定义数据结构、约束和数据库交互，通过Django ORM与数据库进行通信。

示例：（聊天记录模型）

class ChatSession(models.Model):

id = models.CharField(max\_length=36, primary\_key=True, default=uuid.uuid4, editable=False)

user\_id = models.CharField(max\_length=36)

title = models.CharField(max\_length=200, default="新对话")

create\_at = models.DateTimeField(auto\_now\_add=True)

update\_at = models.DateTimeField(auto\_now=True)

is\_active = models.BooleanField(default=True)

chat\_count = models.IntegerField(default=0)

ai\_type = models.IntegerField(default=1)

kb\_id = models.IntegerField(default=0)

3.2 View层设计

在Django中，View层承担了传统MVC架构中Controller的职责，负责处理HTTP请求、执行业务逻辑并返回响应。本项目使用Django REST Framework构建RESTful API。

示例：（创建聊天记录视图）

class CreateChatSessionView(APIView):

def post(self, request):

serializer = ChatSessionSerializer(data=request.data)

if serializer.is\_valid():

chat\_session = serializer.save()

return Response({

'status': 'success',

'message': '会话创建成功',

'data': {

'session\_id': chat\_session.id,

'user\_id': chat\_session.user\_id,

# 其他字段...

}

}, status=status.HTTP\_201\_CREATED)

return Response({

'status': 'error',

'message': '会话创建失败',

'errors': serializer.errors

}, status=status.HTTP\_400\_BAD\_REQUEST)

3.3 URL配置（URLconf）

Django使用URLconf系统来路由HTTP请求到对应的视图函数或类。这是Django处理请求的第一道关卡，在MVT架构中起到连接用户请求和View层的关键作用。

示例：（创建聊天对话的url）

from django.urls import path

from .views import CreateChatSessionView

urlpatterns = [

path('sessions/create/', CreateChatSessionView.as\_view(), name='create\_chat\_session'),

]

4. Django特有的组件和机制

4.1 序列化器（Serializers）

Django REST Framework的序列化器负责数据的序列化和反序列化，以及请求数据的验证，是Django构建API的重要组件。

4.2 权限系统（Permissions）

项目使用了Django REST Framework的权限系统，如IsAuthenticated权限类，控制API的访问权限：

permission\_classes = [IsAuthenticated]

4.3 中间件（Middleware）

Django的中间件处理请求和响应的钩子框架，项目配置了多个中间件：

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

# 其他中间件...

]

4.4 ORM系统

Django的ORM（对象关系映射）系统允许开发者使用Python代码而不是SQL来操作数据库，提供了高度抽象的数据访问接口。

4.5 事务管理

项目使用了Django的事务管理装饰器@transaction.atomic来确保数据库操作的原子性。

5. 应用模块设计

5.1 Account应用

负责用户认证和个人资料管理，定义了自定义用户模型：

AUTH\_USER\_MODEL = 'account.User'

AUTH\_PROFILE\_MODEL = 'account.Profile'

5.2 Knowledge应用

负责知识库管理、文档上传与处理、Markdown内容存储等功能。

5.3 Chat应用

负责聊天会话管理、消息处理和AI交互功能。

6. 数据库配置

项目使用MySQL数据库，通过Django的DATABASES设置进行配置：

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'knowledge\_assistant',

'USER': 'root',

'PASSWORD': 'wyq20050725',

'HOST': 'localhost',

'PORT': '3306',

'OPTIONS': {

'charset': 'utf8mb4',

'use\_unicode': True,

}

}

}

7. 数据流设计

7.1 请求处理流程

用户发送HTTP请求

Django的URLconf系统路由请求到对应的View

View处理请求（验证权限、获取数据、执行业务逻辑）

View调用Model进行数据操作

View返回HTTP响应（通常是JSON格式）

7.2 文档处理流程

用户上传文档到DocumentUploadView

系统验证用户权限和文档有效性

系统保存文档到本地文件系统

系统创建Document记录到数据库

系统可以异步处理文档内容（如切分、索引等）

8. 技术栈总结

Web框架：Django 5.2.5

API框架：Django REST Framework

数据库：MySQL

认证：Django认证系统 + Token认证

前端交互：RESTful API（JSON数据格式）

文件存储：本地文件系统

跨域处理：django-cors-headers

9. Django架构相较于其他架构的优势

9.1 开发效率与迭代速度优势

快速原型开发：相比Java等重型架构，Django的"电池已包含"设计理念使开发人员能够快速构建和验证功能原型

简洁的语法特性：Python语法简洁易读，代码量通常为Java的1/3到1/2，提高开发效率

内置管理后台：Django Admin自动生成功能完善的管理界面，无需额外开发

热重载支持：开发过程中代码修改可实时生效，显著提升开发体验

9.2 AI/大模型集成优势

Python生态无缝集成：作为Python框架，可直接调用OpenAI、Hugging Face等Python SDK，无需额外语言绑定

数据处理能力强：Python丰富的数据处理库（Pandas、NumPy等）便于大模型调用前的数据准备和后处理

向量计算支持：项目中已实现向量存储功能（如test\_vector\_store.py所示），Python在向量计算和相似度搜索方面有丰富库支持

RAG实现便捷：结合知识库功能，Django可快速实现基于检索增强生成的大模型应用

9.3 部署与运维优势

轻量级特性：应用容器镜像小，部署速度快，资源占用低

环境配置简单：相比Java复杂的环境配置，Python环境配置更简单直接

异步处理能力：支持ASGI和协程模型，适合处理大模型调用等I/O密集型操作

启动速度快：开发服务器秒级启动，提高开发和测试效率

9.4 架构设计优势

MVT架构清晰：Model-View-Template架构职责分离明确，代码组织合理

内置ORM系统：比Java的JPA/Hibernate等框架更加简洁直观，减少数据库操作代码量

灵活的URL路由：URLconf系统提供清晰的URL路由定义，易于维护

中间件机制：可快速集成日志、监控等功能，便于大模型调用的调试和优化

这些优势使Django架构特别适合知识密集型AI应用的快速开发和部署，在需要灵活调整、快速迭代的大模型应用场景中表现尤为出色。

10. 扩展与优化建议

性能优化：对频繁查询的模型添加索引，考虑使用缓存机制

异步处理：文档处理和AI交互等耗时操作可使用Django Channels或Celery实现异步处理

安全性增强：生产环境关闭DEBUG模式，配置更严格的CORS策略

数据库优化：考虑使用连接池，优化复杂查询

部署优化：使用Gunicorn/uWSGI配合Nginx部署，提高并发处理能力

3 市面上主流知识库功能对比及未来展望

3.1 市面上主流知识库功能

当前市面上的智能知识库助手，其核心目标是让信息和知识变得更容易获取、更高效利用。它们已经超越了简单的文件存储，更像是一个能理解你、能对话的“第二大脑”。

根据它们侧重的技术和应用场景，主要的功能可以归纳为以下几个方面。

# 二、 市面上主流知识库功能对比及未来展望

## 1、 市面上主流知识库功能

当前市面上的智能知识库助手，其核心目标是让信息和知识变得更容易获取、更高效利用。它们已经超越了简单的文件存储，更像是一个能理解你、能对话的“第二大脑”。

根据它们侧重的技术和应用场景，主要的功能可以归纳为以下几个方面。



## 2、我们的知识库完成的功能

对标市面上主流知识库功能我们的知识库目前已尽做到了，支持多格式文档转化、智能分类与标签化、自然语言问答、多轮对话与上下文理解、内容自动摘要功能。比起市面上主流的智能知识库我们的知识库还有很大不足，主要在知识协作与流转方面以及安全与管控。目前我们的时间和技术水平不支持多人同时协作修改，对于数据安全方面做到了基本的用户密码哈希加密，和保证用户存储内容不会被其他用户访问但是企业级别的数据安全对我们来说还是过于艰巨。目前我们的主要还是偏向文档的智能管理和检索，以及ai对话，但是对文档编辑功能还有待开发，未来我们将继续投入时间开发智能知识库的其他扩展功能，使它可以向市面上的其他知识库助手看齐。

## 3、未来展望

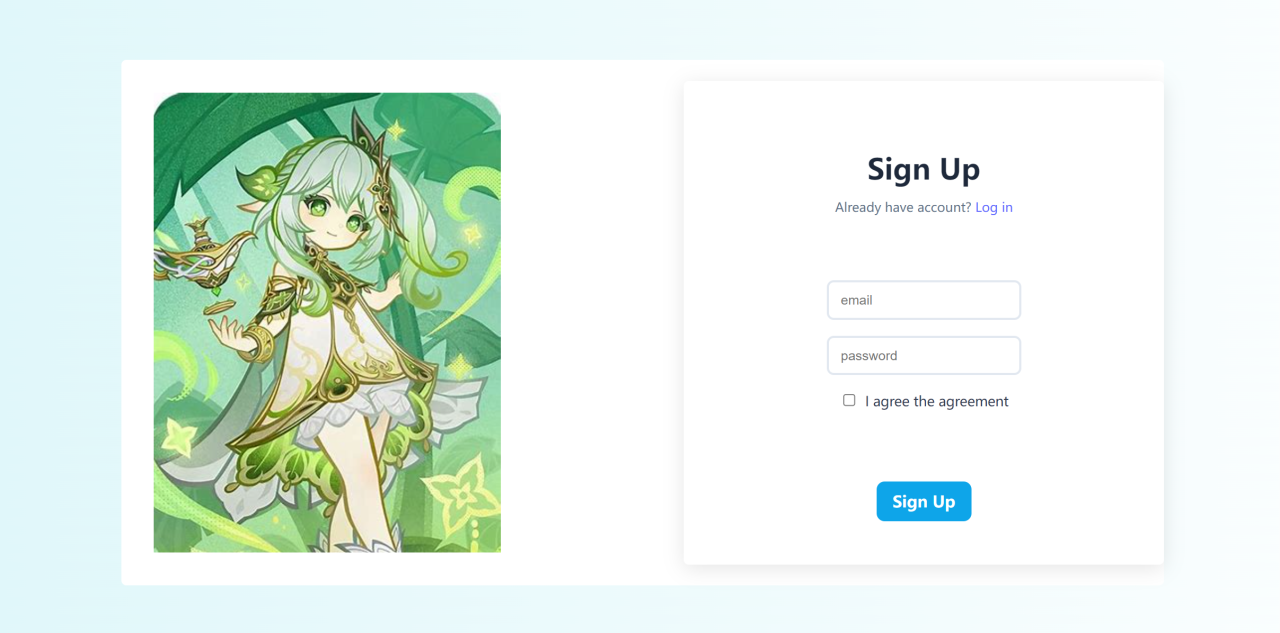
对于后续的开发需求，我们准备先加入文档编辑功能，在文档的增删之后实现文档的编辑，在完成单人文档编辑后，我们希望完成多人文档编辑。多人文档编辑最先可以设置临界区，规定同一时间只支持单人编辑，后续可以仿照github支持多人编辑的同时需要用户手动合并冲突部分，实现文件的版本管理。再后续，希望可以和市面上主流文档管理助手例如飞书看齐，支持多人同时在线编辑，且不需要手动合并内容。与此同时多人协作也需要权限管理相关需求开发的跟进，以保证多人协作之后的文档安全问题。

在大模型方面，我们希望未来我们的智能知识库助手可以支持更多的大模型例如豆包、kimi等等给用户更多的选择。

第七部分 用户界面设计

### 一、桌面布局设计

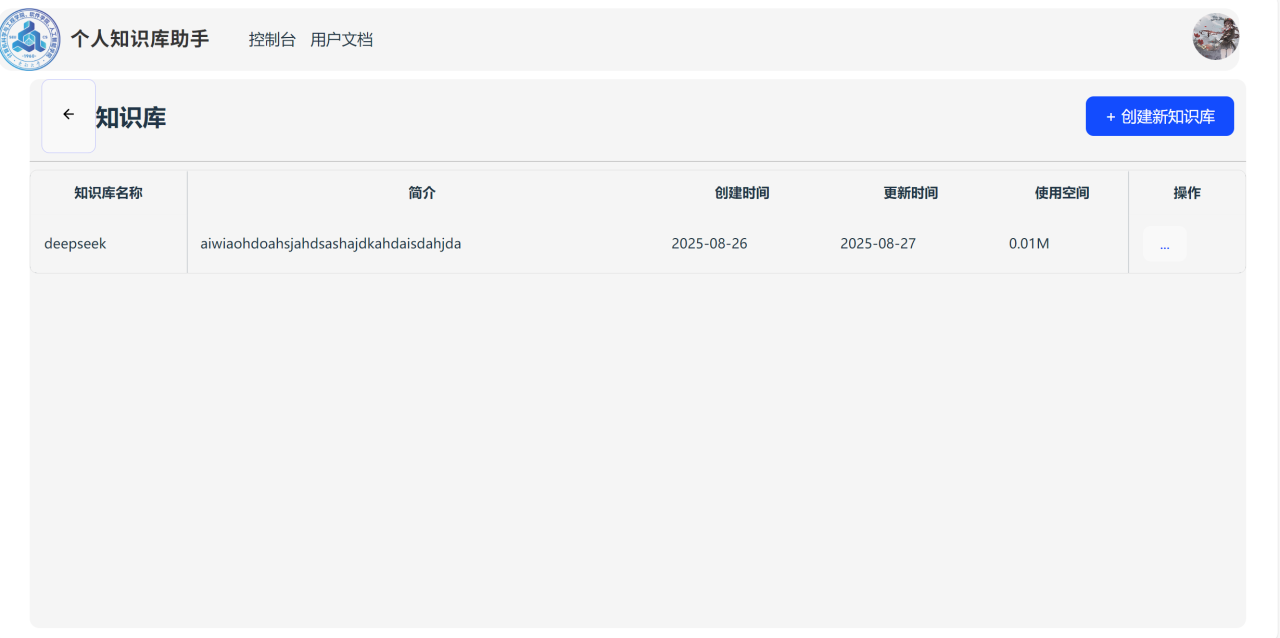
#### 1、登录界面风格



蓝白黑简单配色，简洁明了

简约设计，符合现时代人工智能应用常见审美

#### 2、主桌面风格

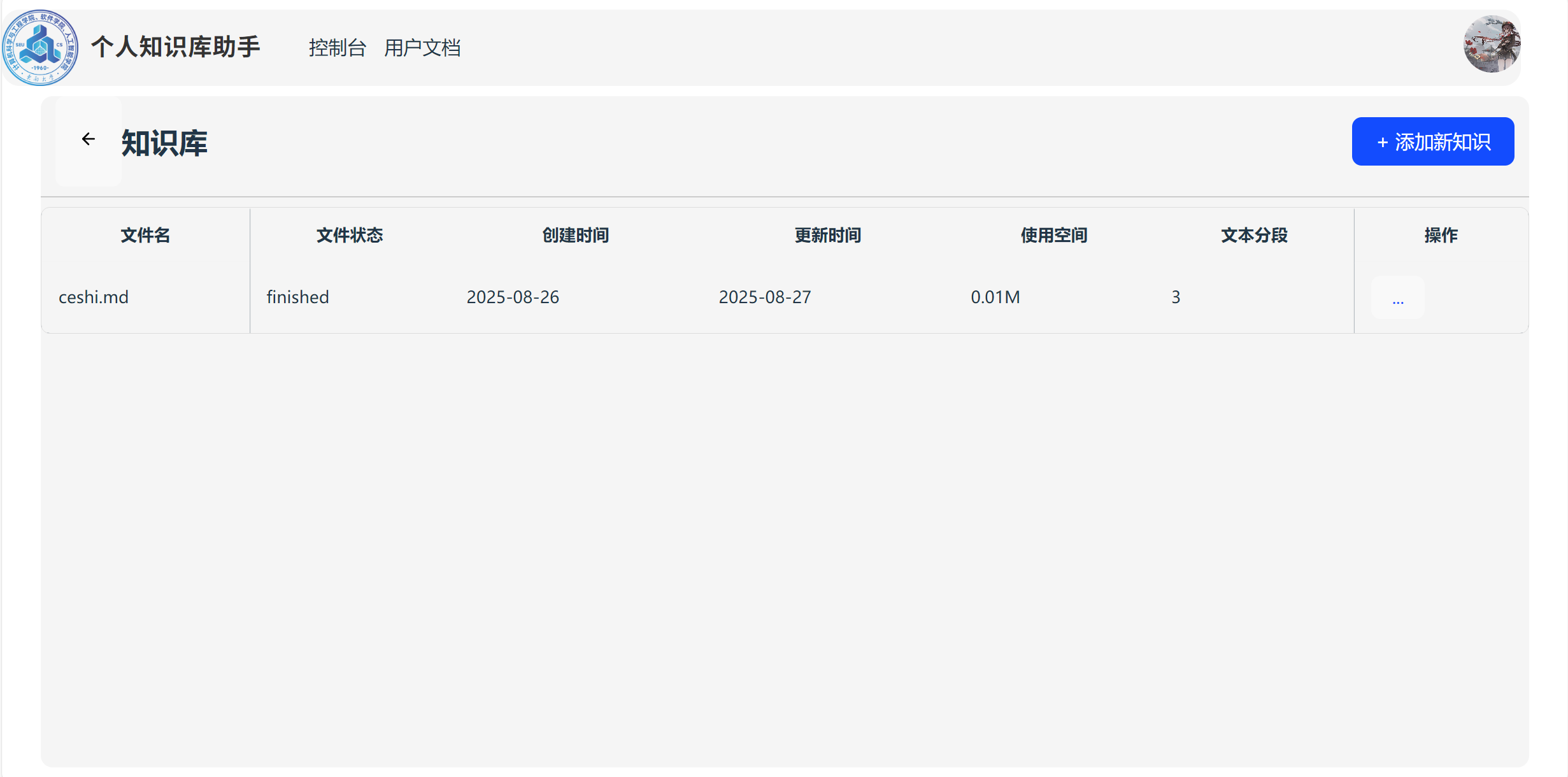


1. 点击创建知识库可新增个人知识库内容。
2. 点击知识库名称可以设置知识库具体数据，包括名称。嵌入维度。
3. 主界面详细展现知识库的创建人、占据空间、创建时间、类型，便于管理个人的多个数据库。

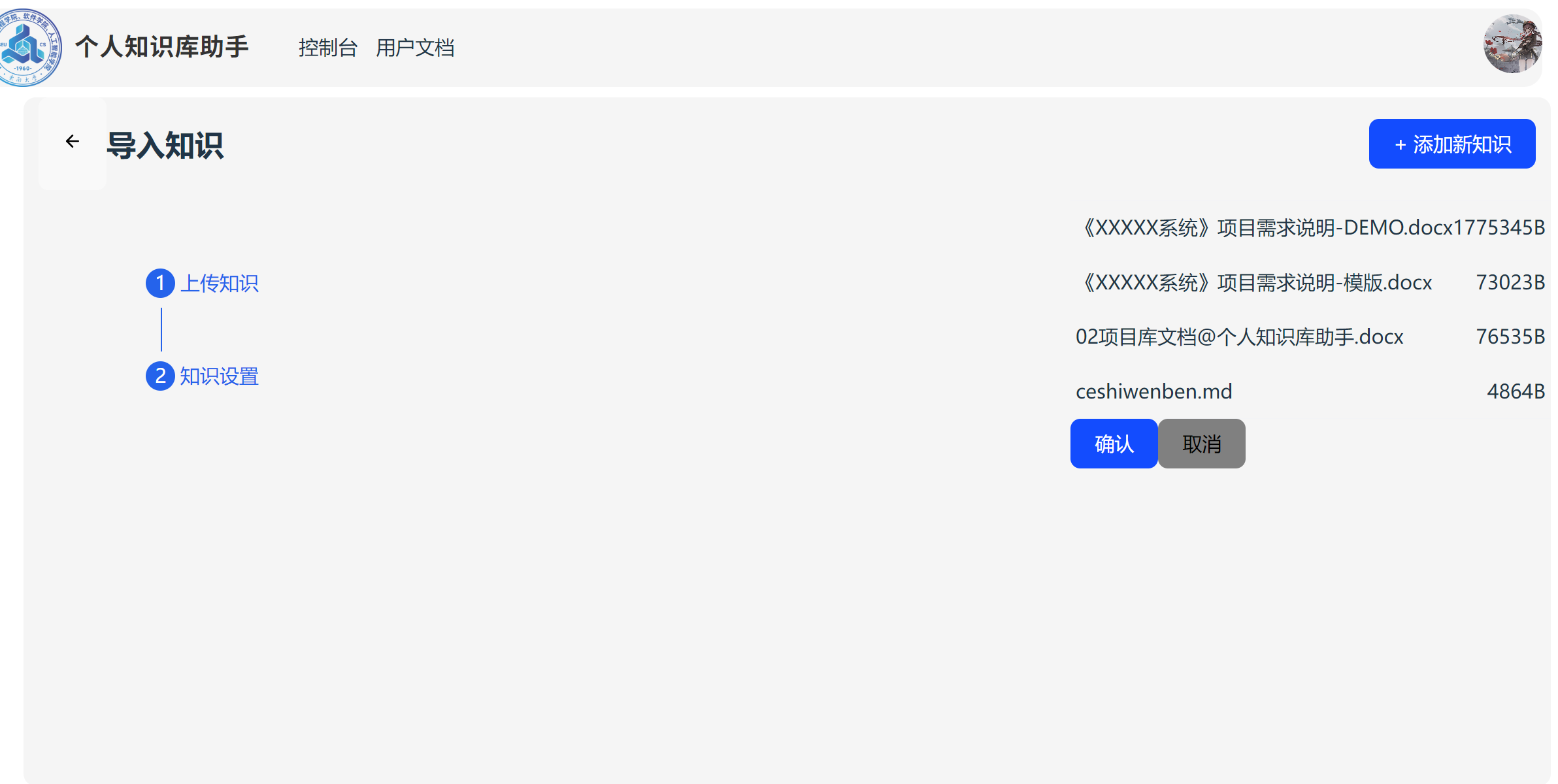
### 二、业务界面风格展示

#### 1、风格展示









统一风格依旧采用蓝白简约风格。理论上大体分成两个部分：知识上传区与知识库设置区。

根据业务不同，每个人可以自由选择知识库的范围与知识库的切片与embedding维度，简化个人需求，同时提供chunk预览界面，清晰明了，辅助用户进行方案决策

第八部分 运行环境和部署

### 一、运行环境

#### 1、客户机器环境

1）硬件​

CPU：Intel i5 9代或同等性能及以上

内存：最低 4 GB RAM，建议 8 GB

​2）操作系统​

Windows 10 或 Windows 11

​3）浏览器​

Chrome 90+ / Firefox 88+ / Edge 90+

#### 2、开发环境要求

|  |  |  |
| --- | --- | --- |
| 项目 | 名称 | 版本 |
| 开发平台 | Windows | 23H2 |
| 开发工具 | VsCode，pycharm | 1.98 |
| 代码管理工具 | github |  |
| 开发环境 | Python | 3.12.3 |

### 二、系统性能要求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| # | 项目 | 模块 | 级别 | 技术参数 |
| 1 | 设计实现技术指标 | 系统架构 | A | 采用C/S模式三层架构。 |
| 2 | 面向对象开发语言与框架 | A | 采用Object-C2.0开发语言和CoreData,Quarz2D等技术。采用Coreplot开源图表绘制框架 |
| 3 | 注释和文档 | A | 符合CMMI软件开发过程标准文档（至少提供：需求、概要、详细设计、测试报告、部署和环境、用户手册），代码注释量>=30%。 |
| 4 | 模块化和适合实训 | A | SOA设计、模块化，保证系统各模块单元较强的独立性适合实训教学。 |
| 5 | 测试覆盖率 | A | 功能覆盖率>=100%，业务覆盖率>=100%，语言覆盖率>=100%，逻辑覆盖率>=80%。 |
| 6 | 资源利用率要求 | CPU占用率 | B | <=50%利用率（附近标准配置） |
| 7 | 内存使用率 | B | <=75%利用率（附近标准配置） |
| 8 | 响应时间要求 | 服务器 | B | <=100ms（附近标准配置） |
| 9 | 网络 | B | <=100ms（附近标准配置） |
| 10 | 客户端 | B | <=5s（附近标准配置） |
| 11 | 系统稳定性要求 | 成熟性 | A | 真实的用户，成功使用本系统 |
| 12 | 稳定性 | B | 无故障运行时间>=365天，系统恢复时间<=2小时。 |
| 13 | 先进性 | A | 采用目前体验最好、最流行的iPhone与iPad移动终端。 |
| 14 | 典型意义 | A | 案例项目要有典型意义，有推广价值。 |
| 15 | Web服务接口要求 | WebService服务器端 | A | 支持协议定制，支持安全过滤，支持消息队列，多种模式重发，支持成功检测。消息延迟<=2S，丢包<=0.001%。 |
| 16 | WebService移动终端 | A | 支持协议定制，支持安全过滤，支持消息队列，多种模式重发，支持成功检测。消息延迟<=2S，丢包<=0.001%。 |
| 17 | MAP平台端 | A | 消息延迟<=2S，丢包<=0.001%。 |
| 18 | 集成部署环境 | 服务器 | A | Microsoft Windows XP/2003 Server |
| 19 | 数据库 | A | Oracle DB 10g 及以上 |
| 20 | 手持机 | A | IPhone 4/iOS 4.0以上 |
| 21 | PC电脑 | A | Microsoft Windows XP 及以上 |
|  |  |  |  |  |
| 说明：级别（A:表示非常重要必须达到的技术性能要求,B:表示重要推荐达到的技术性能要求,C：表示非重要可以弱化的技术性能要求.） | | | | |