产品需求文档 (PRD)

方案 修改一下 第一版 前后端数据库部署在一台服务器 用docker隔离 使用

- Redis：存储会话数据。
- PostgreSQL：存储用户信息、翻译历史、文件元数据。
- 本地目录：存储 PDF 文件。

项目概述

项目名称： AI助手Web应用

版本： v1.0

创建日期： 2025-11-08

文档作者： 产品团队

---

1. 项目背景与目标

1.1 项目背景

开发一个精简版的AI助手Web应用，提供核心的AI交互功能，包括智能对话、文本翻译和PDF文档分析。

1.2 项目目标

- 构建一个用户友好的AI助手平台
- 提供稳定、高效的AI交互体验
- 实现三大核心功能：聊天、翻译、ChatPDF

1.3 目标用户

- 需要AI辅助工作的专业人士
- 学习研究人员
- 需要文档分析和翻译的用户

---

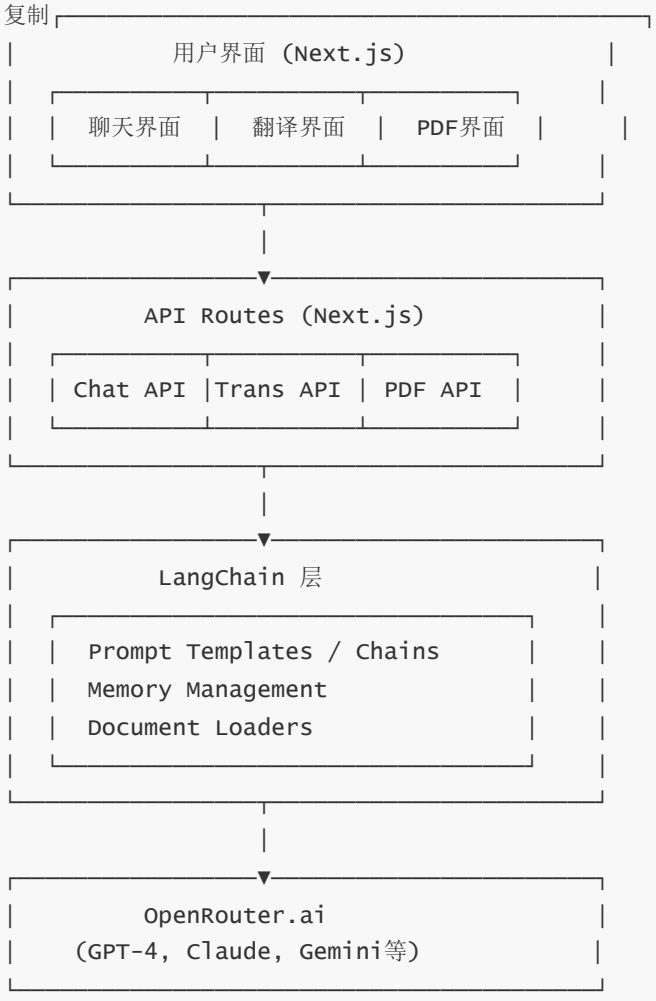
2. 技术架构

2.1 技术栈选型

层级	技术选型	说明
前端框架	Next.js 14+	使用App Router，支持SSR/SSG
后端框架	Next.js API Routes	统一技术栈，简化部署
AI编排	LangChain	处理与大模型的交互逻辑

层级	技术选型	说明
模型接口	OpenRouter.ai	统一的模型调用接口
样式方案	Tailwind CSS	快速构建响应式UI
状态管理	Zustand / React Context	轻量级状态管理
数据库	PostgreSQL + Prisma	存储用户数据和对话历史
文件存储	本地存储 / S3	PDF文件临时存储

2.2 系统架构图



3. 功能需求

3.1 核心功能模块

3.1.1 智能对话 (Chat)

功能描述： 用户可以与AI进行多轮对话，支持上下文记忆。

功能点：

- ✅ 实时流式输出（SSE）
- ✅ 对话历史记录
- ✅ 多会话管理（创建、删除、重命名）
- ✅ 模型选择（GPT-4, Claude, Gemini等）
- ✅ 上下文记忆（可配置记忆轮数）
- ✅ Markdown渲染（代码高亮、表格等）
- ✅ 复制、重新生成功能

用户流程：

```
1. 用户进入聊天界面
2. 选择或创建新会话
3. 输入问题/指令
4. 系统流式返回AI回复
5. 用户可继续追问或开启新话题
```

API设计：

```
POST /api/chat
Request: {
  message: string,
  conversationId: string,
  model: string,
  stream: boolean
}
Response: {
  id: string,
  content: string,
  role: 'assistant',
  timestamp: number
}
```

3.1.2 文本翻译 (Translation)

功能描述： 提供多语言互译功能，支持常见语言对。

功能点：

- ✅ 自动检测源语言
- ✅ 支持语言：中文、英文、日文、韩文、法文、德文、西班牙文等
- ✅ 双向翻译
- ✅ 批量文本翻译

- ✅ 翻译历史记录
- ✅ 一键复制译文

用户流程：

```
1. 用户进入翻译界面
2. 输入待翻译文本
3. 选择目标语言（可选，支持自动检测）
4. 点击翻译按钮
5. 显示翻译结果
6. 可复制或继续翻译
```

API设计：

```
POST /api/translate
Request: {
  text: string,
  sourceLang?: string,  // 可选，自动检测
  targetLang: string
}
Response: {
  translatedText: string,
  detectedLang: string,
  targetLang: string
}
```

3.1.3 PDF文档分析 (ChatPDF)

功能描述： 上传PDF文档后，用户可以对文档内容进行提问和分析。

功能点：

- ✅ PDF文件上传（支持拖拽）
- ✅ 文档内容解析和向量化
- ✅ 基于文档内容的问答
- ✅ 引用来源标注（页码）
- ✅ 多文档管理
- ✅ 文档摘要生成
- ✅ 支持文件大小限制（如10MB）

用户流程：

```
1. 用户进入ChatPDF界面
2. 上传PDF文件
3. 系统解析文档并向量化存储
4. 用户对文档提问
5. AI基于文档内容回答并标注来源
6. 可继续追问或上传新文档
```

技术实现：

- 使用 `pdf-parse` 或 `pdfjs-dist` 解析PDF
- 使用 LangChain 的 `RecursiveCharacterTextSplitter` 分割文本
- 使用 OpenAI Embeddings 向量化
- 使用向量数据库（如 Pinecone / Chroma）存储
- 使用 LangChain 的 `RetrievalQA` 链

API设计：

```
POST /api/pdf/upload
Request: FormData (file)
Response: {
  documentId: string,
  filename: string,
  pageCount: number,
  status: 'processing' | 'ready'
}

POST /api/pdf/chat
Request: {
  documentId: string,
  message: string
}
Response: {
  answer: string,
  sources: Array<{page: number, content: string}>
}
```

3.2 辅助功能

3.2.1 用户认证（可选MVP阶段）

- 简单的邮箱/密码登录
- 或使用 NextAuth.js 集成第三方登录

3.2.2 会话管理

- 侧边栏展示历史会话
- 会话搜索
- 会话导出（Markdown格式）

---

4. 界面设计

4.1 整体布局

4.2 页面结构

4.2.1 聊天页面

- 左侧：会话列表（可折叠）
- 中间：对话内容区（消息气泡）
- 底部：输入框 + 发送按钮 + 模型选择

4.2.2 翻译页面

- 左侧：源文本输入框
- 右侧：译文显示区
- 中间：语言选择器 + 翻译按钮

4.2.3 ChatPDF页面

- 左侧：已上传文档列表
- 中间：对话区（类似聊天界面）
- 右侧：PDF预览（可选）
- 顶部：上传按钮

---

5. 数据模型设计

5.1 数据库表结构（Prisma Schema）

```
// 用户表
model User {
  id            String         @id @default(cuid())
  email         String         @unique
  name          String?
  createdAt     DateTime       @default(now())
  conversations Conversation[]
  documents     Document[]
}

// 会话表
model Conversation {
  id        String    @id @default(cuid())
  title     String
  type      String    // 'chat' | 'translate' | 'pdf'
  userId    String
  user      User      @relation(fields: [userId], references: [id])
  messages  Message[]
  createdAt DateTime  @default(now())
  updatedAt DateTime  @updatedAt
}

// 消息表
model Message {
  id             String       @id @default(cuid())
  content        String       @db.Text
  role           String       // 'user' | 'assistant' | 'system'
  conversationId String
  conversation   Conversation @relation(fields: [conversationId], references: [id])
  createdAt      DateTime     @default(now())
}

// 文档表
model Document {
  id          String   @id @default(cuid())
  filename    String
  filepath    String
  pageCount   Int
  status      String   // 'processing' | 'ready' | 'error'
  userId      String
  user        User     @relation(fields: [userId], references: [id])
  vectorStoreId String?
  createdAt   DateTime @default(now())
}
```

6. 开发计划

6.1 开发阶段划分

Phase 1: 项目初始化（1-2天）

- ✅ Next.js项目搭建
- ✅ 配置Tailwind CSS
- ✅ 配置Prisma + PostgreSQL
- ✅ 基础UI组件开发（Button, Input, Card等）
- ✅ 路由结构设计

Phase 2: 聊天功能开发（3-4天）

- ✅ 聊天界面开发
- ✅ 集成LangChain + OpenRouter
- ✅ 实现流式输出
- ✅ 会话管理功能
- ✅ 对话历史存储

Phase 3: 翻译功能开发（2-3天）

- ✅ 翻译界面开发
- ✅ 实现翻译API
- ✅ 语言检测功能
- ✅ 翻译历史记录

Phase 4: ChatPDF功能开发（4-5天）

- ✅ PDF上传功能
- ✅ PDF解析和向量化
- ✅ 向量数据库集成
- ✅ 基于检索的问答实现
- ✅ 文档管理界面

Phase 5: 优化与测试（2-3天）

- ✅ 性能优化
- ✅ 错误处理
- ✅ 单元测试
- ✅ UI/UX优化
- ✅ 响应式适配

Phase 6: 部署上线（1-2天）

- ✅ 环境配置
- ✅ Vercel部署
- ✅ 数据库迁移
- ✅ 监控配置

总计：约15-20天

---

7. 技术实现细节

7.1 LangChain集成示例

```
复制// lib/langchain/chat.ts
import { ChatOpenAI } from "langchain/chat_models/openai";
import { HumanMessage, AIMessage } from "langchain/schema";

export async function createChatChain(apiKey: string, model: string) {
  const chat = new ChatOpenAI({
    openAIApiKey: apiKey,
    modelName: model,
    streaming: true,
    configuration: {
      baseURL: "https://openrouter.ai/api/v1",
    },
  });

  return chat;
}
```

7.2 OpenRouter配置

```
复制// lib/openrouter.ts
export const OPENROUTER_CONFIG = {
  baseURL: "https://openrouter.ai/api/v1",
  models: {
    gpt4: "openai/gpt-4-turbo",
    claude: "anthropic/claude-3-opus",
    gemini: "google/gemini-pro",
  },
};
```

7.3 环境变量配置

```
复制# .env.local
DATABASE_URL="postgresql://..."
OPENROUTER_API_KEY="sk-or-..."
NEXTAUTH_SECRET="..."
NEXTAUTH_URL="http://localhost:3000"
```

---

8. 非功能性需求

8.1 性能要求

- 页面首次加载时间 < 2秒
- API响应时间 < 3秒（不含AI生成时间）
- 支持流式输出，提升用户体验
- PDF处理时间 < 30秒（10MB文件）

8.2 安全要求

- API Key加密存储
- 用户数据隔离
- 文件上传类型和大小限制
- XSS和CSRF防护
- Rate Limiting（防止滥用）

8.3 可扩展性

- 模块化设计，易于添加新功能
- 支持多模型切换
- 支持自定义Prompt模板

---

9. 风险与挑战

风险项	影响	应对策略
OpenRouter API稳定性	高	实现重试机制、降级方案
PDF解析准确性	中	多种解析库备选、人工校验
向量数据库成本	中	使用本地Chroma或限制文档数量
并发处理能力	中	实现队列机制、限流
Token消耗成本	高	实现用量统计、设置配额

---

10. 成功指标

- ✅ 三大核心功能完整可用
- ✅ 界面响应流畅，无明显卡顿
- ✅ AI回复准确率 > 85%
- ✅ 系统稳定性 > 99%
- ✅ 用户满意度 > 4/5

---

11. 附录

11.1 参考资源

- Next.js官方文档: https://nextjs.org/docs
- LangChain文档: https://js.langchain.com/docs
- OpenRouter文档: https://openrouter.ai/docs

11.2 相关文档

- 技术设计文档（TDD）
- API接口文档
- 测试用例文档
- 部署运维文档

文档状态： 待评审

下一步行动：

1. 技术团队评审可行性
2. UI/UX团队提供设计稿
3. 确定开发排期
4. 启动Phase 1开发