Mini-OpenClaw 开发需求文档 (PRD)

一、项目介绍

1. 功能与目标定位

Mini-OpenClaw 是一个基于 **Python** 重构的、轻量级且高度透明的 AI Agent 系统，旨在复刻并优化 OpenClaw（原名 Moltbot/Clawdbot）的核心体验。

本项目不追求构建庞大的 SaaS 平台，而是致力于打造一个**运行在本地的、拥有"真实记忆"的数字副手**。其核心差异化定位在于：

- **文件即记忆 (File-first Memory)**：摒弃不透明的向量数据库，回归最原始、最通用的 Markdown/JSON 文件系统。用户的每一次对话、Agent 的每一次反思，都以人类可读的文件形式存在。
- **技能即插件 (Skills as Plugins)**：遵循 Anthropic 的 Agent Skills 范式，通过文件夹结构管理能力，实现"拖入即用"的技能扩展。
- **透明可控**：所有的 System Prompt 拼接逻辑、工具调用过程、记忆读写操作对开发者完全透明，拒绝"黑盒"Agent。

2. 项目核心技术架构

本项目要求完全采用 **前后端分离** 架构，后端作为纯 API 服务运行。

- **后端语言**：Python 3.10+ (强制使用 Type Hinting)。
- **Web 框架**：**FastAPI** (提供 RESTful 接口，支持异步处理)。
- **Agent 编排引擎**：**LangChain 1.x (Stable Release)**。
  - **核心 API**：必须使用 `create_agent` API (`from langchain.agents import create_agent`)。这是 LangChain 1.0 版本发布的最新标准 API，用于构建基于 Graph 运行时的 Agent。
  - **核心说明**：严禁使用旧版的 `AgentExecutor` 或早期的 `create_react_agent` （旧链式结构）。`create_agent` 底层虽然基于 LangGraph 运行时，但提供了更简洁的标准化接口，本项目应紧跟这一最新范式。
- **RAG 检索引擎**：**LlamaIndex (LlamaIndex Core)**。
  - 用于处理非结构化文档的混合检索（Hybrid Search），作为 Agent 的知识外挂。
- **模型接口**：兼容 OpenAI API 格式（支持 OpenRouter, DeepSeek, Claude 等模型直连）。
- **数据存储**：本地文件系统 (Local File System) 为主，不引入 MySQL/Redis 等重型依赖。

二、内置工具

Mini-OpenClaw 在启动时，除了加载用户自定义的 Skills 外，必须内置以下 5 个核心基础工具（Core Tools）。根据"优先使用 LangChain 原生工具"的原则，技术选型更新如下：

1. 命令行操作工具 (Command Line Interface)

- **功能描述**：允许 Agent 在受限的安全环境下执行 Shell 命令。
- **实现逻辑**：
  - **直接使用 LangChain 内置工具**：`langchain_community.tools.ShellTool`。
  - **配置要求**：
    - 初始化时需配置 `root_dir` 限制操作范围（沙箱化），防止 Agent 修改系统关键文件。
    - 需预置黑名单拦截高危指令（如 `rm -rf /`）。
- **工具名称**：`terminal`。

2. Python 代码解释器 (Python REPL)

- **功能描述**：赋予 Agent 逻辑计算、数据处理和脚本执行的能力。
- **实现逻辑**：
  - **直接使用 LangChain 内置工具**：`langchain_experimental.tools.PythonREPLTool`。
  - **配置要求**：
    - 该工具会自动创建一个临时的 Python 交互环境。
    - **注意**：由于 `PythonREPLTool` 位于 `experimental` 包中，需确保依赖项安装正确。
- **工具名称**：`python_repl`。

3. Fetch 网络信息获取

- **功能描述**：用于获取指定 URL 的网页内容，Agent 联网的核心。
- **实现逻辑**：
  - **直接使用 LangChain 内置工具**：`langchain_community.tools.RequestsGetTool`。
  - **增强配置 (Wrapper)**：
    - 原生 `RequestsGetTool` 返回的是原始 HTML，Token 消耗巨大。
    - **必须封装**：建议继承该类或创建一个 Wrapper，在获取内容后使用 `BeautifulSoup` 或 `html2text` 库清洗数据，仅返回 Markdown 或纯文本内容。
- **工具名称**：`fetch_url`。

4. 文件读取工具 (File Reader)

- **功能描述**：用于精准读取本地指定文件的内容。这是 Agent Skills 机制的核心依赖，用于读取 `SKILL.md` 的详细说明。
- **实现逻辑**：
  - **直接使用 LangChain 内置工具**：`langchain_community.tools.file_management.ReadFileTool`。
  - **配置要求**：
    - 必须设置 `root_dir` 为项目根目录，严禁 Agent 读取项目以外的系统文件。
- **工具名称**：`read_file`。

5. RAG 检索工具 (Hybrid Retrieval)

- **功能描述**：当用户询问具体的知识库内容（非对话历史）时，Agent 可调用此工具进行深度检索。
- **技术选型**：**LlamaIndex**。
- **实现逻辑**：
  - **索引构建**：支持扫描指定目录（如 `knowledge/`）下的 PDF/MD/TXT 文件，构建本地索引。
  - **混合检索**：必须实现 **Hybrid Search**（关键词检索 BM25 + 向量检索 Vector Search）。
  - **持久化**：索引文件需持久化存储在本地（`storage/`）。
- **工具名称**：`search_knowledge_base`。

三、mini OpenClaw 的 Agent Skills 系统

1. Agent Skills 基础功能介绍

mini OpenClaw 的 Agent Skills 遵循 **"Instruction-following" (指令遵循)** 范式，而非传统的 "Function-calling" (函数调用) 范式。这意味着 Skills 本质上是**教会 Agent 如何使用基础工具（如 Python/Terminal）去完成任务的说明书**，而不是预先写好的 Python 函数。

Agent Skills 以文件夹形式存在于 `backend/skills/` 目录下。

2. Agent Skills 载入与执行流程

2.1 Agent Skills 读取流程 (Bootstrap)

在 Agent 启动或会话开始时，系统扫描 `skills` 文件夹，读取每个 `SKILL.md` 的元数据（Frontmatter），并将其汇总生成 `SKILLS_SNAPSHOT.md`。

`SKILLS_SNAPSHOT.md` 示例：

```
<available_skills>
  <skill>
    <name>get_weather</name>
    <description>获取指定城市的实时天气信息</description>
    <location>./backend/skills/get_weather/SKILL.md</location>
  </skill>
</available_skills>
```

注意：`location` 使用相对路径。

2.2 Agent Skills 调用流程 (Execution)

这是本系统最独特的地方：

1. **感知**：Agent 在 System Prompt 中看到 `available_skills` 列表。
2. **决策**：当用户请求“查询北京天气”时，Agent 发现 `get_weather` 技能匹配。
3. **行动 (Tool Call)**：Agent **不调用** `get_weather()` 函数（因为它不存在），而是调用 `read_file(path="./backend/skills/get_weather/SKILL.md")`。
4. **学习与执行**：Agent 读取 Markdown 内容，理解操作步骤（例如：“使用 fetch_url 访问某天气 API” 或 “使用 python_repl 运行以下代码”），然后**动态调用 Core Tools** (Terminal/Python) 来完成任务。

四、mini OpenClaw 对话记忆管理系统设计

1. 本地优先原则

所有记忆文件（Markdown/JSON）均存储在本地文件系统，确保完全的数据主权和可解释性。

2. 系统提示词 (System Prompt) 构成

System Prompt 由以下 6 部分动态拼接而成（按顺序）：

1. `SKILLS_SNAPSHOT.md` (能力列表)
2. `SOUL.md` (核心设定)
3. `IDENTITY.md` (自我认知)
4. `USER.md` (用户画像)
5. `AGENTS.md` (行为准则 & **记忆操作指南**)
6. `MEMORY.md` (长期记忆)

截断策略：如果拼接后 Token 超出模型限制（或单文件超 20k 字符），需对超长部分进行截断并在末尾添加 `...[truncated]` 标识。

3. AGENTS.md 的默认配置 (核心修正)

由于 Agent 默认并不知道它是通过"阅读文件"来学习技能的，因此必须在初始化时生成一个包含明确指令的 `AGENTS.md`。

- **必须包含的元指令 (Meta-Instructions)**:

```
# 操作指南

## 技能调用协议 (SKILL PROTOCOL)
你拥有一个技能列表 (SKILLS_SNAPSHOT)，其中列出了你可以使用的能力及其定义文件的位置。
**当你要使用某个技能时，必须严格遵守以下步骤：**
1. 你的第一步行动永远是使用 `read_file` 工具读取该技能对应的 `location` 路径下的 Markdown 文件。
2. 仔细阅读文件中的内容、步骤和示例。
3. 根据文件中的指示，结合你内置的 Core Tools (terminal, python_repl, fetch_url) 来执行具体任务。
**禁止**直接猜测技能的参数或用法，必须先读取文件！

## 记忆协议
...
```

4. 会话存储 (Sessions)

- **路径**：`backend/sessions/{session_name}.json`
- **格式**：标准 JSON 数组，包含 `user`, `assistant`, `tool` (function calls) 类型的完整消息记录。

五、后端 API 接口规范 (FastAPI)

后端服务作为独立进程运行，负责 Agent 逻辑、文件读写和状态管理。

- **服务端口**：`8002`

- **基础 URL**：`http://localhost:8002`

1. 核心对话接口

- **Endpoint**: `POST /api/chat`
- **功能**: 发送用户消息，获取 Agent 回复。
- **Request**:

```
{
  "message": "查询一下北京的天气",
  "session_id": "main_session",
  "stream": true
}
```

- **Response**: 支持 **SSE (Server-Sent Events)** 流式输出，实时推送 Agent 的思考过程 (Thought/Tool Calls) 和最终回复。

2. 文件管理接口 (用于前端编辑器)

- **Endpoint**: `GET /api/files`
  - **Query**: `path=memory/MEMORY.md`
  - **功能**: 读取指定文件的内容。
- **Endpoint**: `POST /api/files`
  - **Body**: `{ "path": "...", "content": "..." }`
  - **功能**: 保存对 Memory 或 Skill 文件的修改。

3. 会话管理接口

- **Endpoint**: `GET /api/sessions`
  - **功能**: 获取所有历史会话列表。

六、前端开发要求

1. 设计理念与布局架构

前端采用 **IDE（集成开发环境）风格**，三栏式布局。

- **左侧 (Sidebar)**：导航 (Chat/Memory/Skills) + 会话列表。
- **中间 (Stage)**：对话流 + **思考链可视化** (Collapsible Thoughts)。
- **右侧 (Inspector)**：Monaco Editor，用于实时查看/编辑正在使用的 `SKILL.md` 或 `MEMORY.md`。

2. 技术栈

- **框架**: Next.js 14+ (App Router), TypeScript
- **UI**: Shadcn/UI, Tailwind CSS, Lucide Icons
- **Editor**: Monaco Editor (配置为 Light Theme)

3. UI/UX 风格规范

- **色调**: **浅色 Apple 风格 (Frosty Glass)。**
  - 背景：纯白/极浅灰 (`#fafafa`)，高透毛玻璃效果。
  - 强调色：**克莱因蓝** (Klein Blue) 或 **活力橙。**
- **导航栏**: 顶部固定，半透明。
  - 左中：**"mini OpenClaw"**
  - 右侧：**"赋范空间"** (链接至 `https://fufan.ai`)。

七、项目目录结构参考

建议 Claude Code 按照以下结构进行初始化：

```
mini-openclaw/
├── backend/                # FastAPI + LangChain/LangGraph
│   ├── app.py              # 入口文件 (Port 8002)
│   ├── memory/             # 记忆存储
│   │   ├── logs/           # Daily logs
│   │   └── MEMORY.md       # Core memory
│   ├── sessions/           # JSON 会话记录
│   ├── skills/             # Agent Skills 文件夹
│   │   └── get_weather/
│   │       └── SKILL.md
│   ├── workspace/          # System Prompts (SOUL.md, etc.)
│   ├── tools/              # Core Tools 实现
│   ├── graph/              # LangGraph 状态机定义
│   └── requirements.txt
│
├── frontend/               # Next.js 14+
│   ├── src/
│   │   ├── app/
│   │   ├── components/
│   │   │   ├── chat/       # 聊天组件
│   │   │   └── editor/     # Monaco Wrapper
│   │   └── lib/
│   │       └── api.ts      # Fetch wrapper for port 8002
│   └── package.json
│
└── README.md
```