项目开发手册：AI 驱动网络小说创作 SaaS 平台

版本: v1.0.0 **状态**: 待开发 **文档目标**: 为前后端开发人员提供统一的技术规范、数据库设计及功能实现标准。

1. 技术栈规范 (Tech Stack)

所有开发工作必须严格遵守以下技术选型：

1.1 前端 (Frontend)

- **框架**: React 18 + TypeScript + Vite
- **路由**: React Router v6 (使用 Data API 模式)
- **状态管理**: Zustand (用于全局 App 状态), TanStack Query (用于服务器数据缓存)
- **UI 组件库**: Shadcn UI + Tailwind CSS
- **核心功能库**:
  - 富文本编辑器: **Tiptap** (Headless Editor)
  - 流程图/关系图: **React Flow**
  - 拖拽排序: **dnd-kit**
  - 图标: **Lucide React**
- **HTTP 客户端**: Axios (需封装拦截器)

1.2 后端 (Backend)

- **框架**: FastAPI (Python 3.10+)
- **数据库**: PostgreSQL (生产环境) / SQLite (开发环境)
- **ORM**: SQLModel (结合 SQLAlchemy + Pydantic)
- **认证**: OAuth2 + JWT (JSON Web Tokens)
- **AI 集成**: OpenAI SDK (兼容 DeepSeek/Moonshot API)
- **数据迁移**: Alembic

2. 前后端通信规范 (API Interface Standards)

2.1 约定

- **接口前缀**: `/api/v1`
- **请求格式**: `Content-Type: application/json`
- **鉴权方式**: HTTP Header -> `Authorization: Bearer <token>`
- **命名风格**:
  - API URL: 连字符 (kebab-case), 例 `/api/v1/user-profile`
  - JSON Request (前端发给后端): **蛇形命名 (snake_case)** (为了配合 Python Pydantic)
  - JSON Response (后端发给前端): **蛇形命名 (snake_case)** (前端需定义 interface 适配)

2.2 响应结构 (Response Wrapper)

除了流式 (Streaming) 接口外，所有 JSON 响应必须遵循统一格式：

```
// 成功 (200 OK)
{
  "code": 200,
  "message": "Success",
  "data": { ... } // 具体数据
}

// 失败 (4xx, 5xx)
{
  "code": 4001, // 业务错误码
  "message": "用户名已存在",
  "detail": "..." // 可选调试信息
}
```

3. 数据库设计 (Database Schema)

后端开发需严格按照以下 SQLModel 定义建表。

3.1 用户系统 (User System)

Table: `users` | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | Primary Key, Auto Increment | | `email` | String | Yes | Unique, Index | | `hashed_password` | String | Yes | Bycrpt 加密后的哈希值 | | `nickname` | String | No | 用户昵称 | | `avatar_url` | String | No | 头像链接 | | `is_active` | Boolean | Yes | Default: True | | `created_at` | DateTime | Yes | 注册时间 |

3.2 小说核心 (Novel Core)

Table: `novels` | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | PK | | `user_id` | Integer | Yes | FK -> users.id (级联删除) | | `title` | String | Yes | 书名 | | `description` | Text | No | 简介/大纲 | | `cover_url` | String | No | 封面图 | | `status` | String | Yes | enum: serializing(连载), finished(完结) | | `created_at` | DateTime | Yes | |

Table: `volumes` (卷) | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | PK | | `novel_id` | Integer | Yes | FK -> novels.id | | `title` | String | Yes | 卷名，如“第一卷：初出茅庐” | | `order_index` | Integer | Yes | 排序索引，用于拖拽排序 |

Table: `chapters` (章) | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | PK | | `volume_id` | Integer | Yes | FK -> volumes.id | | `title` | String | Yes | 章节名 | | `content` | Text | No | 章节正文 (HTML/JSON string from Tiptap) | | `summary` | Text | No | AI 生成的本章摘要 | | `word_count` | Integer | Yes | 字数统计 | | `order_index` | Integer | Yes | 排序索引 |

3.3 世界观与人物 (World Building)

Table: `characters` | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | PK | | `novel_id` | Integer | Yes | FK -> novels.id | | `name` | String | Yes | 姓名 | | `role` | String | Yes | enum: protagonist(主角), antagonist(反派), supporting(配角) | | `tags` | String | No | JSON 字符串，如 ["高冷", "剑客"] | | `bio` | Text | No | 人物小传/Prompt 上下文源 | | `avatar` | String | No | 人物立绘 |

Table: `relationships` (关系图) | 字段名 | 类型 | 必填 | 说明 | | :--- | :--- | :--- | :--- | | `id` | Integer | Yes | PK | | `novel_id` | Integer | Yes | FK -> novels.id (方便查询整本书关系) | | `source_id` | Integer | Yes | FK -> characters.id (起点人物) | | `target_id` | Integer | Yes | FK -> characters.id (终点人物) | | `relation_type` | String | Yes | 关系类型 (如: 师徒, 仇敌) | | `description` | String | No | 关系描述 |

4. API 接口清单 (API Endpoints)

4.1 认证模块 (Auth)

- `POST /api/v1/auth/register`：注册 (Body: email, password)
- `POST /api/v1/auth/login`：登录 (Body: email, password) -> 返回 access_token
- `GET /api/v1/auth/me`：获取当前用户信息 (Require Auth)

4.2 小说管理模块 (Novel)

- `GET /api/v1/novels`：获取当前用户的所有书籍列表
- `POST /api/v1/novels`：创建新书
- `GET /api/v1/novels/{id}/toc`：**关键接口**。获取书籍目录树 (包含所有卷和章节，嵌套结构，用于前端 Sidebar 渲染)

4.3 章节管理模块 (Chapter)

- `GET /api/v1/chapters/{id}`：获取单章详情（含正文）
- `PUT /api/v1/chapters/{id}`：保存章节内容 (自动计算字数)
- `POST /api/v1/chapters`：新建章节

4.4 世界观模块 (World)

- `GET /api/v1/novels/{id}/characters`：获取某书所有人物
- `GET /api/v1/novels/{id}/graph`：获取关系图数据 (返回 Nodes 和 Edges 格式)

4.5 AI 助手模块 (AI Copilot)

- `POST /api/v1/ai/polish`：**流式接口 (Streaming)**
  - Body:
    - `text`：选中的原文
    - `instruction`：指令 (如: "扩写", "润色")
    - `context_characters`：[ID List] (选中的相关人物 ID)
  - Response: Server-Sent Events (SSE) 文本流。

5. 前端功能与页面规划 (Frontend Specs)

5.1 路由结构 (Routes)

- `/` (Landing Page): 官网首页
- `/login`：登录页
- `/register`：注册页

- `/dashboard` : (Private) 书架页面，显示项目卡片
- `/editor/:novelId` : (Private) 编辑器主布局
  - 子路由 `/editor/:novelId/chapter/:chapterId` : 写作界面
  - 子路由 `/editor/:novelId/world` : 人物关系图界面

5.2 核心组件细节

A. 侧边栏 (Sidebar) - `EditorLayout`

- **功能:** 显示“卷 -> 章”的树形结构。
- **交互:** 支持拖拽改变章节顺序 (调用后端 API 更新 order_index)。
- **状态:** 选中章节高亮。

B. 写作区 (Writer) - `TiptapEditor`

- **库:** `@tiptap/react` , `@tiptap/starter-kit`
- **功能:**
  - 基础格式 (Bold, Italic, Heading)。
  - **AI 浮动菜单 (Bubble Menu):** 选中文字后弹出 -> 点击“AI 润色” -> 调用流式 API -> 文本实时替换。
  - **自动保存:** 使用 `useDebounce` 钩子，停止打字 2 秒后自动触发 PUT 保存接口。

C. 人物关系图 (Graph) - `RelationshipView`

- **库:** `reactflow`
- **逻辑:**
  - 从后端 fetch `characters` 转换为 `Nodes` 。
  - 从后端 fetch `relationships` 转换为 `Edges` 。
  - 支持点击连线，弹出 Shadcn Dialog 编辑关系描述。

6. 开发环境搭建指南

后端 (Backend)

```
# 1. 创建虚拟环境
python -m venv venv
source venv/bin/activate

# 2. 安装依赖
pip install fastapi "uvicorn[standard]" sqlmodel psycopg2-binary pyjwt passlib[bcrypt]

# 3. 运行
uvicorn main:app --reload
```

前端 (Frontend)

```
# 1. 初始化
npm create vite@latest client -- --template react-ts

# 2. 安装核心库
npm install axios zustand @tanstack/react-query react-router-dom lucide-react clsx tail
npm install @tiptap/react @tiptap/starter-kit
npm install reactflow

# 3. 运行
npm run dev
```

7. 交付标准 (Definition of Done)

1. **用户隔离**: 用户 B 登录后无法通过 API 访问用户 A 的 `novel_id`（后端需做权限校验）。
2. **数据完整**: 刷新页面后，小说目录结构、章节内容、人物关系图不丢失。
3. **AI 体验**: 点击润色后，文字必须是“打字机”式逐字出现，而不是等待 10 秒一次性出现。