---

### \*\*结构化技术需求文档：单项目看板\*\*

#### \*\*1. 项目/功能概述 (Overview)\*\*

开发一个单页面的项目管理看板应用。用户可以在一个固定的项目看板上，通过可视化的卡片来创建、查看、更新和删除任务，并能在不同状态列之间拖拽任务。

#### \*\*2. 核心功能点 (Core Features)\*\*

\* \*\*固定看板视图：\*\* 页面核心是一个看板，包含四个固定的列：“未开始”、“设计中”、“开发中”、“已上线”。

\* \*\*任务创建：\*\* 在每个列的顶部有一个“+”按钮，点击后会显示一个局部表单，用于创建新任务。

\* \*\*任务属性：\*\* 每个任务包含以下信息：

\* 标题 (Title)

\* 优先级 (Priority)：一个下拉选择器，选项为 P0, P1, P2。

\* 负责人 (Assignee)：一个简单的文本输入框。

\* \*\*任务状态管理：\*\* 任务的状态由其所在的列决定。

\* \*\*拖拽移动：\*\* 用户可以通过鼠标拖拽任务卡片，将其从一个列移动到另一个列，从而改变任务状态。

\* \*\*行内编辑：\*\* 用户可以直接点击任务卡片上的标题或负责人文本，文本会变为输入框，修改后即可保存。

\* \*\*任务删除：\*\* 每个任务卡片上都有一个删除按钮。

#### \*\*3. 技术规格 (Technical Specifications)\*\*

\*\*前端 (Frontend):\*\*

\* \*\*页面/组件 (Pages/Components):\*\*

\* `KanbanBoard.vue`: 主页面组件，容纳整个看板，负责获取所有任务数据并分发。

\* `KanbanColumn.vue`: 列组件，接收一个状态（如“未开始”）和对应的任务列表作为属性。它将管理该列下的所有任务卡片和拖拽区域。

\* `TaskCard.vue`: 任务卡片组件，显示单个任务的详细信息（标题、优先级、负责人），并处理行内编辑和删除逻辑。

\* `NewTaskForm.vue`: 新建任务的表单组件，点击“+”时显示，用于输入新任务的信息。

\* \*\*用户流程 (User Flow):\*\*

1. \*\*加载页面：\*\* 用户打开应用，`KanbanBoard` 组件加载，并从后端获取所有任务数据。

2. \*\*渲染看板：\*\* 任务数据根据其 `status` 字段被分配到对应的 `KanbanColumn` 中，并渲染成 `TaskCard`。

3. \*\*创建任务：\*\* 用户点击某列的“+”按钮，`NewTaskForm` 出现。填写信息并提交后，数据被发送到后端。成功后，新的 `TaskCard` 出现在该列。

4. \*\*修改状态：\*\* 用户鼠标按住一个 `TaskCard`，将其拖拽到另一个 `KanbanColumn` 并松开。前端捕获此事件，向后端发送更新任务状态的请求。

5. \*\*编辑任务：\*\* 用户点击 `TaskCard` 上的标题，标题文本变成输入框。修改后按回车或失焦，前端向后端发送更新任务标题的请求。

6. \*\*删除任务：\*\* 用户点击 `TaskCard` 上的删除图标，前端弹出确认提示后，向后端发送删除任务的请求，成功后从界面上移除该卡片。

\* \*\*数据交互 (Data Interaction):\*\*

\* \*\*获取任务:\*\* 页面加载时，向后端发送 `GET /api/tasks` 请求。

\* \*\*创建任务:\*\* 提交新任务表单时，向后端发送 `POST /api/tasks` 请求。

\* \*\*更新任务:\*\*

\* 拖拽任务改变状态时，发送 `PUT /api/tasks/{id}` 请求，请求体中包含新的 `status`。

\* 行内编辑保存时，发送 `PUT /api/tasks/{id}` 请求，请求体中包含更新后的字段（如 `title` 或 `assignee`）。

\* \*\*删除任务:\*\* 点击删除按钮时，发送 `DELETE /api/tasks/{id}` 请求。

\*\*后端 (Backend):\*\*

\* \*\*API 接口 (API Endpoints):\*\*

\* `GET /api/tasks`

\* \*\*描述:\*\* 获取所有任务列表。

\* \*\*响应 (JSON):\*\* `[ { "id": 1, "title": "任务一", "priority": "P0", "status": "开发中", "assignee": "张三" }, ... ]`

\* `POST /api/tasks`

\* \*\*描述:\*\* 创建一个新任务。

\* \*\*请求体 (JSON):\*\* `{ "title": "新任务标题", "priority": "P1", "assignee": "李四", "status": "未开始" }`

\* \*\*响应 (JSON):\*\* `{ "id": 2, "title": "新任务标题", ... }` (返回创建成功后的任务对象)

\* `PUT /api/tasks/{id}`

\* \*\*描述:\*\* 更新一个已存在的任务（可用于更新任何字段）。

\* \*\*请求体 (JSON):\*\* `{ "status": "已上线" }` 或 `{ "title": "修改后的标题" }`

\* \*\*响应 (JSON):\*\* `{ "id": 1, "title": "修改后的标题", ... }` (返回更新后的完整任务对象)

\* `DELETE /api/tasks/{id}`

\* \*\*描述:\*\* 根据 ID 删除一个任务。

\* \*\*响应:\*\* 状态码 `204 No Content` 或 `{ "message": "删除成功" }`

\* \*\*数据模型 (Data Models):\*\*

\* \*\*任务表 (Tasks Table):\*\*

\* `id` (主键, 整数, 自动增长)

\* `title` (字符串, 不能为空)

\* `priority` (字符串, 建议值为 'P0', 'P1', 'P2')

\* `status` (字符串, 建议值为 '未开始', '设计中', '开发中', '已上线')

\* `assignee` (字符串, 可以为空)

\* `created\_at` (时间戳, 记录创建时间)

\* `updated\_at` (时间戳, 记录最后更新时间)

#### \*\*4. 技术栈建议 (Tech Stack Suggestion)\*\*

\* \*\*前端 (Frontend):\*\*

\* \*\*框架:\*\* \*\*Vue.js\*\* (对初学者友好，社区活跃，文档清晰)。

\* \*\*拖拽库:\*\* \*\*`vue.draggable.next`\*\* (专门为 Vue 3 设计的拖拽库，简单易用)。

\* \*\*样式:\*\* \*\*Tailwind CSS\*\* (一个“功能优先”的 CSS 框架，无需写太多自定义 CSS 就能快速构建简洁干净的界面)。

\* \*\*后端 (Backend):\*\*

\* \*\*框架:\*\* \*\*Python Flask\*\* (一个轻量级的“微框架”，代码量少，非常适合入门和构建小型 API)。

\* \*\*数据库:\*\* \*\*SQLite\*\* (一个基于文件的数据库，无需安装独立的数据库服务，非常适合本地开发和小型项目)。

#### \*\*5. 开发步骤建议 (Development Steps)\*\*

我为你把整个过程拆解成了可以一步步执行的小任务，你可以按照这个清单来推进：

1. \*\*步骤一：环境准备\*\*

\* 安装 Node.js (用于前端) 和 Python (用于后端)。

\* 创建项目总文件夹，并在其中分别创建 `frontend` 和 `backend` 两个子文件夹。

2. \*\*步骤二：后端开发 (先让数据跑起来)\*\*

\* 在 `backend` 文件夹中，安装 Flask 和 Flask-SQLAlchemy (用于操作数据库)。

\* 创建 Flask 应用，并配置使用 SQLite 数据库。

\* 定义 `Task` 数据模型。

\* 编写并实现 `GET`, `POST`, `PUT`, `DELETE` 四个 API 接口。

\* 使用 Postman 或 Insomnia 这样的工具测试你的 API，确保它们都能正常工作。

3. \*\*步骤三：前端框架搭建\*\*

\* 在 `frontend` 文件夹中，使用 `npm create vue@latest` 创建一个新的 Vue.js 项目。

\* 按照官方指南为你的 Vue 项目集成 Tailwind CSS。

\* 创建 `KanbanBoard`, `KanbanColumn`, `TaskCard` 这几个组件的空文件，并画出静态的页面布局（先不用管数据）。

4. \*\*步骤四：前端连接后端，显示数据\*\*

\* 在 `KanbanBoard` 组件中，使用 `fetch` 或 `axios` 在页面加载时调用后端的 `GET /api/tasks` 接口。

\* 将获取到的任务数据保存在组件的状态中，并正确地分发给四个 `KanbanColumn`。

\* 让 `TaskCard` 组件能正确显示任务的标题、优先级和负责人。

5. \*\*步骤五：实现核心功能\*\*

\* \*\*创建任务：\*\* 完成 `NewTaskForm` 组件，实现 `POST /api/tasks` 的调用，并在成功后实时更新界面。

\* \*\*拖拽功能：\*\* 在 `KanbanColumn` 中集成 `vue.draggable.next`，实现任务卡片的拖拽。在拖拽结束后，调用 `PUT /api/tasks/{id}` 更新任务状态。

\* \*\*行内编辑：\*\* 在 `TaskCard` 中编写逻辑，实现点击文本切换到输入框，并在保存时调用 `PUT /api/tasks/{id}`。

\* \*\*删除任务：\*\* 在 `TaskCard` 中添加删除按钮，点击后调用 `DELETE /api/tasks/{id}` 并从界面移除卡片。

6. \*\*步骤六：润色和收尾\*\*

\* 添加一些简单的加载提示（比如“正在加载中...”）。

\* 美化卡片和表单的样式，确保整体风格简洁。

\* 处理一些边界情况，例如当某一列没有任何任务时显示提示信息。