TODO List 项目说明文档

1. 技术选型

- **编程语言**： JavaScript

  理由：生态成熟、快速开发。

- **框架/库**：React+Vite

  理由：组件化方便，利于扩展。 Vite热更新快、构建速度快，能够提供快速开发体验。

- **状态管理**：useReducer + useContext

  理由：比起Redux，此方案较为轻量且足够满足待办应用的状态共享需求。比起useState，此方案状态更新逻辑集中、可维护性高。

- **数据库/存储**：LocalStroage

  理由：轻量，浏览器原生支持。

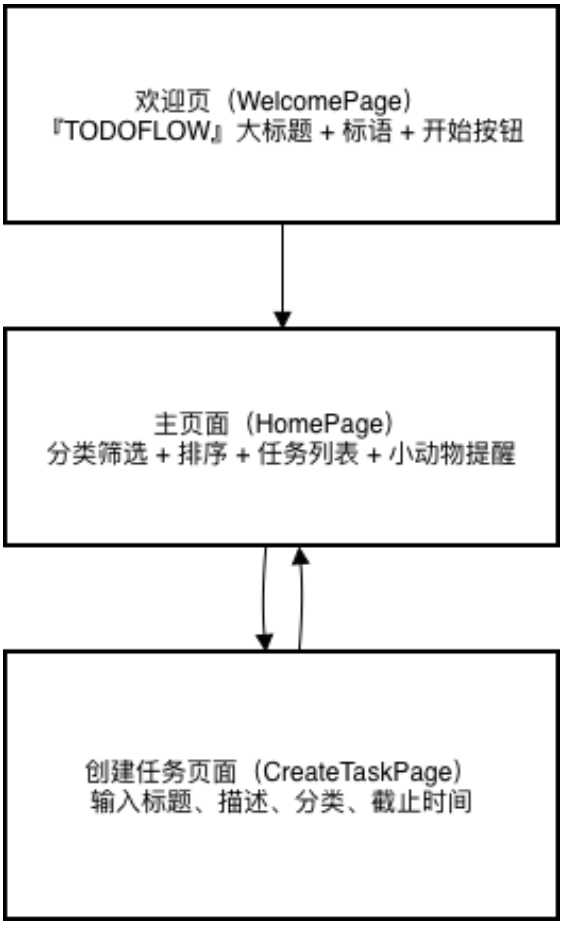
- **替代方案对比**：

  未使用数据库/indexDB：因为本项目无需大量结构化数据，也无需复杂事务逻辑，因此 localStorage 是最简单稳定的选择。

  未使用本地文件：由于浏览器的安全限制，web前端无法直接读写本地文件。

2. 项目结构设计

- 整体架构：按照 “单页应用 + 多视图” 方式构建：



- 目录结构：

```
src/
 ├── App.jsx
 ├── index.css
 ├── Context/
 │    └── TasksContext.jsx    # 全局任务状态与localStorage 持久化
 ├── hooks/
 │    └── useReminder.js      # 提醒逻辑（贪睡机制）
 ├── pages/
 │    ├── StartPage.jsx       # 欢迎页
 │    ├── HomePage.jsx        # 查看任务列表
 │    └── CreateTaskPage.jsx  # 创建任务页面
 ├── Components/
 │    ├── AddTask/            # 增加任务
 │    ├── TaskList/           # 渲染多个 TaskItem
 │    ├── CategoryFilter/     # 提供分类选择
 │    ├── SortBar/            # 展示排序下拉框（按创建时间/优先级/截止时间）
 │    ├── TaskItem/           # 显示单个任务的数据
 │    └── CatAssistant/       # 提醒助手
 ├── utils/
 │    └── dateHelpers.js      # 时间格式化
```

3. 需求细节与决策

3.1 必须功能

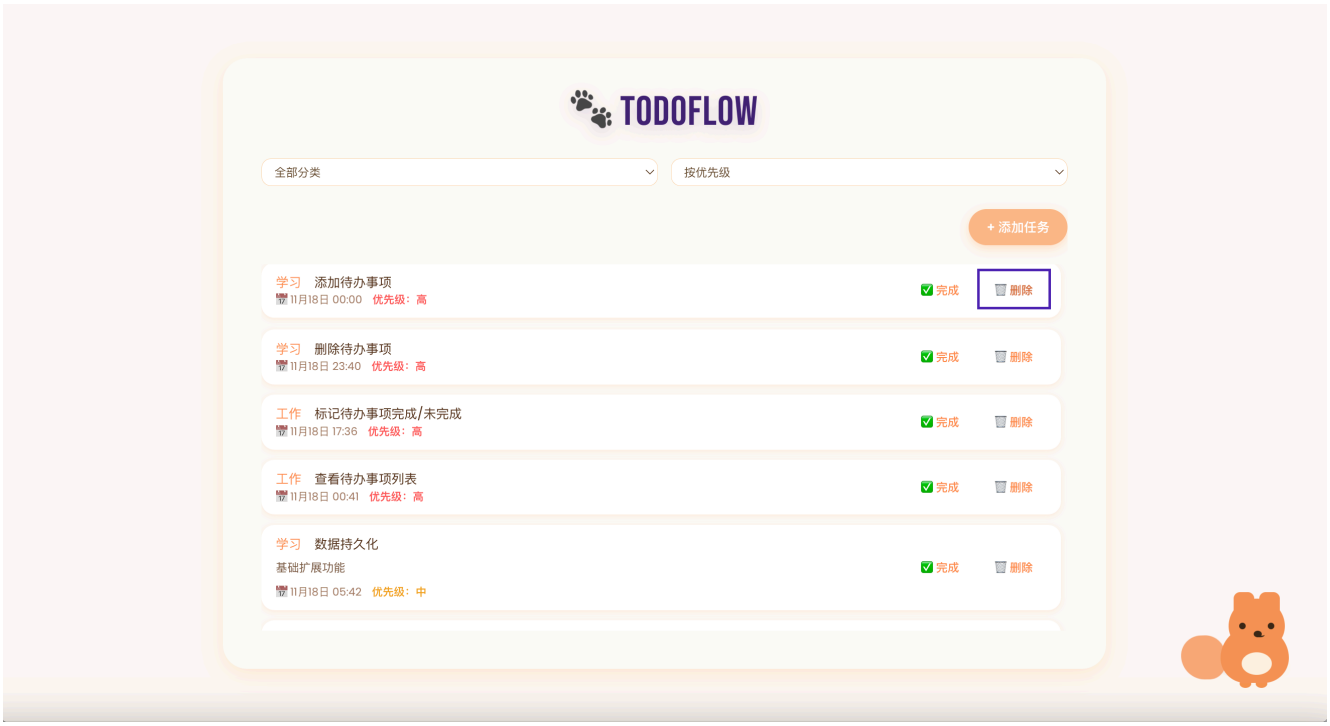
1. **添加待办事项**

   添加待办事项功能允许用户在 *CreateTaskPage* 中创建一条新的任务记录。任务包含标题、分类、优先级、截止时间和描述等字段，其中标题为必填项。用户在填写表单后点击“添加任务”，前端会做简单校验，确认标题不为空，最后跳转回首页展示最新任务列表。

2. 删除待办事项

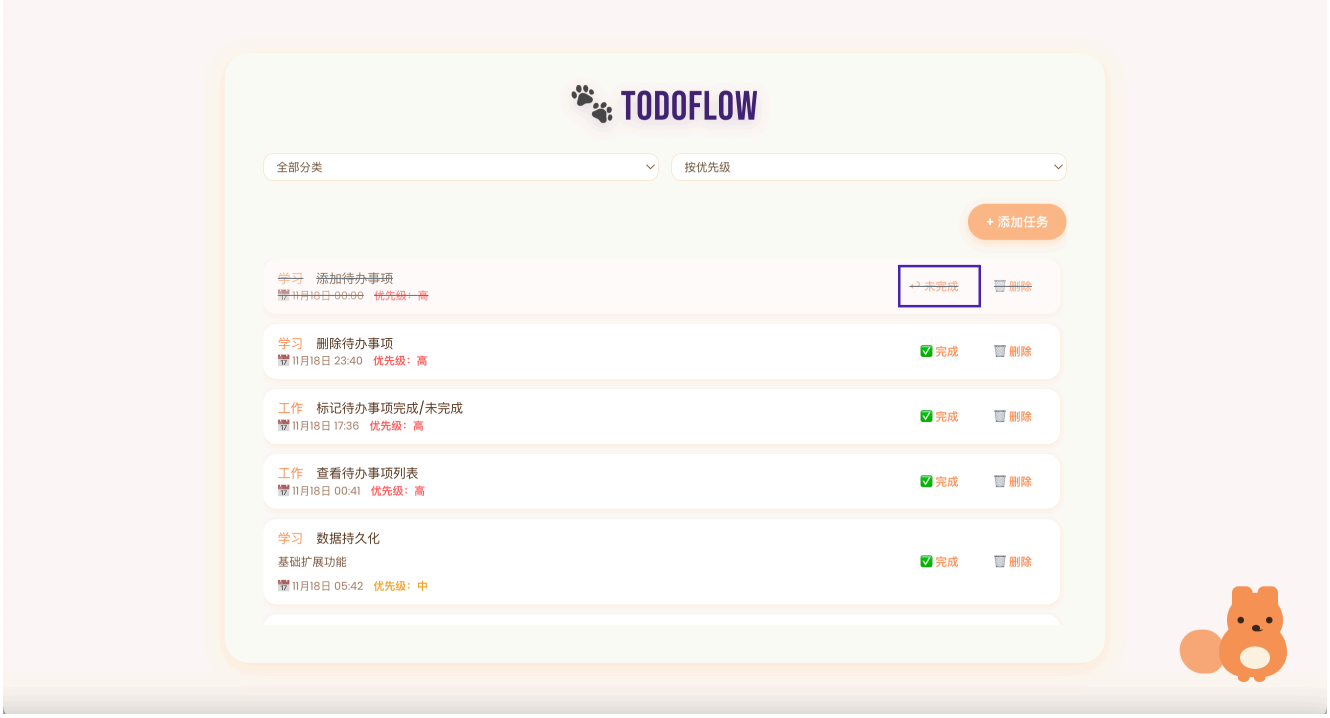
   删除功能允许用户在任务列表中移除不再需要的待办事项。用户在 HomePage 的 TaskItem 上点击“删除”按钮后，会触发 TasksContext 中的 removeTask() 方法，将对应任务从全局任务数组中移除，并同步更新 localStorage，以保持状态持久化。

   在设计上，删除动作是立即生效的，不需要二次确认，从而保证操作快速、直接。为避免数组操作带来的复杂性，系统采用简单的基于 id 的过滤方式更新任务列表，让删除过程稳定、易维护。



3. 标记待办事项完成/未完成

   用户可以在任务列表中切换某条任务的完成状态。点击 TaskItem 的完成按钮后，界面会立即更新展示任务的当前状态，完成任务会变灰并加删除线。交互上保持即时反馈，不需要额外确认。状态变更会调用 TasksContext 中的 `toggleTaskCompletion()` 方法，在全局任务列表中找到对应任务并翻转其 `completed` 字段，同时同步更新 localStorage，确保页面刷新后仍能保留结果。



4. 查看待办事项列表

   系统在首页展示所有用户创建的待办事项列表。页面加载时会从 TasksContext 获取当前的任务数组，该数组已经通过 localStorage 持久化，保证刷新页面后数据仍然存在。TaskList 组件根据任务状态（未完成 / 已完成）自动渲染不同样式，并按默认排序规则展示，如按创建时间从新到旧排列。在交互上，用户无需任何操作即可实时看到新增、更新或删除任务后的最新结果。

3.2 基础扩展功能

1. 数据持久化

   为了保证用户刷新页面或重新打开应用时依旧能够看到之前的任务数据，系统采用 localStorage 作为前端持久化方案。

   所有增删改操作都会通过 TasksContext 同步更新内部状态，并在副作用中实时写入 localStorage，以字符串化 JSON 的形式保存完整任务列表。应用初始化时，会优先尝试从 localStorage 读取已有数据并恢复到全局状态，实现“无后端依赖”的轻量级持久化能力。

2. 任务分类

   任务分类功能允许用户根据不同类别（*工作 / 学习 / 生活*）对任务进行筛选，以便快速查看特定类型的待办事项。系统在任务对象中保存 `category` 字段，并在主列表渲染前根据当前选中的类别进行过滤。

3. 任务排序

   排序功能允许用户按照截止时间、创建时间和优先级进行排序。系统在 TasksContext 中维护一个 `sortBy` 状态，用于记录用户选择的排序字段，并在任务列表渲染时根据该字段动态调整展示顺序：

   - **按创建时间（createdAt）**：让用户能够按照任务生成先后查看，适用于回顾或按习惯管理任务；

- **按截止时间（deadline）**：按照紧急程度排序，使要优先处理的任务自动靠前；
- **按优先级（priority）**：根据任务重要性进行排序，帮助用户从高优先级任务入手。

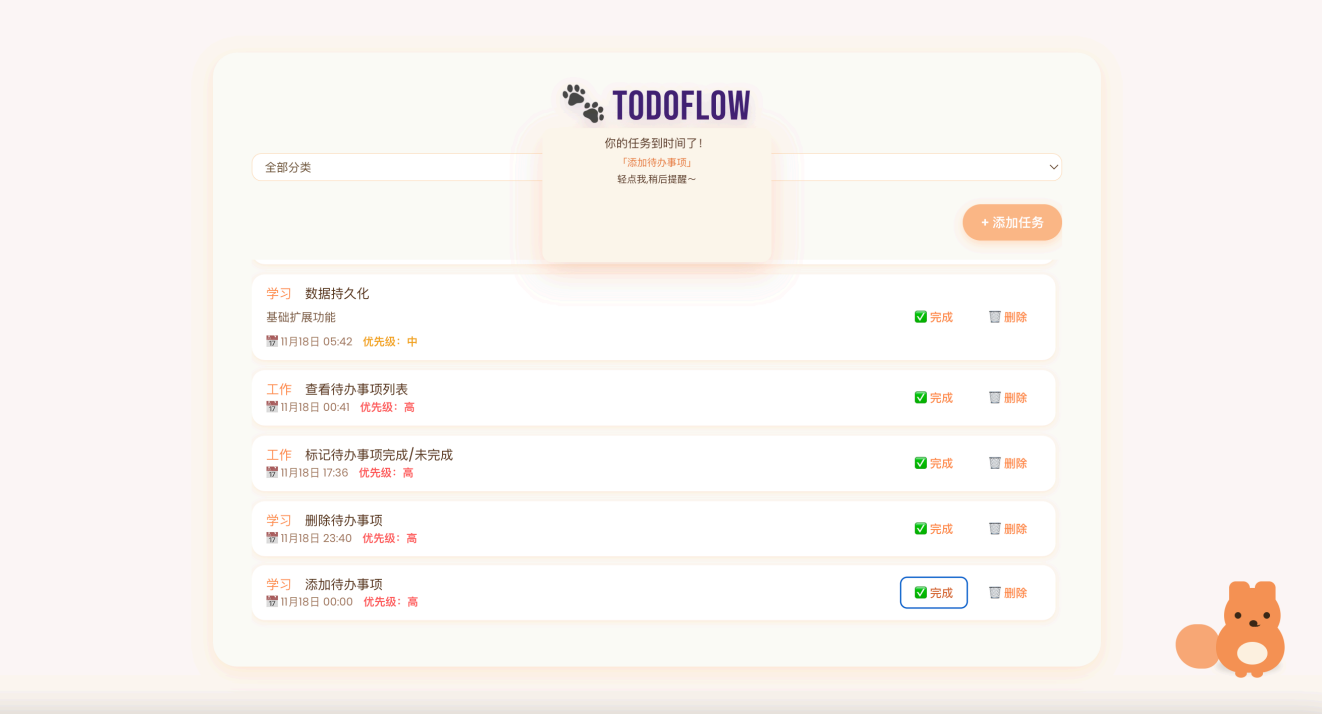
3.3 进阶挑战功能

1. 提醒/通知功能。

   本系统提供轻量级的任务提醒机制，通过前端定时轮询与提醒助手组件结合，实现无需后端即可运行的本地通知能力。

   在底层实现上，`useReminder()` 会以 30 秒为周期扫描未完成且设置了截止时间的任务，通过工具函数将日期转换为精确的时间戳后，与当前时间进行比较。当任务到达截止时间且未被贪睡时，Hook 会将该任务设置为当前提醒对象，触发提醒 UI。为避免频繁打扰用户，系统为每个任务维护一个 `snoozedMap` 记录，如果用户点击提醒助手关闭提醒，该任务将进入 5 分钟的贪睡期，在此期间不会再次触发提醒。

   提醒的呈现由CatAssistant 组件负责。组件在接收到提醒信号时，会自动播放一个轻量的 Web Audio API 提示音，同时展示带有任务标题的提示气泡，引导用户操作。视觉上，小狐狸会从闲置状态切换到警觉动画，吸引用户注意；关闭提醒后则恢复静止与轻微摆尾的休息状态。所有动画与提示气泡均以独立 CSS 动画实现。



4. AI 使用说明

- AI工具：ChatGPT
- 使用 AI 的环节：
  - 需求分析
  - 架构搭建，状态初始化
  - 页面结构初稿生成
  - CSS 绘图与动效设计
  - 提醒功能、任务排序模块代码生成
  - 部分文档编写

- 修改：
    1. 解耦页面设计，对于AI设计的单页结构改成React Router 多页结构
    2. 对于提醒功能，设计贪睡机制
    3. 修改 AI 代码中不合理的 deadline 检测逻辑，将以日为粒度的截止日期改为具体的截止时间

5. 运行与测试方式

- 本地运行方式

```
npm install
npm run dev
```

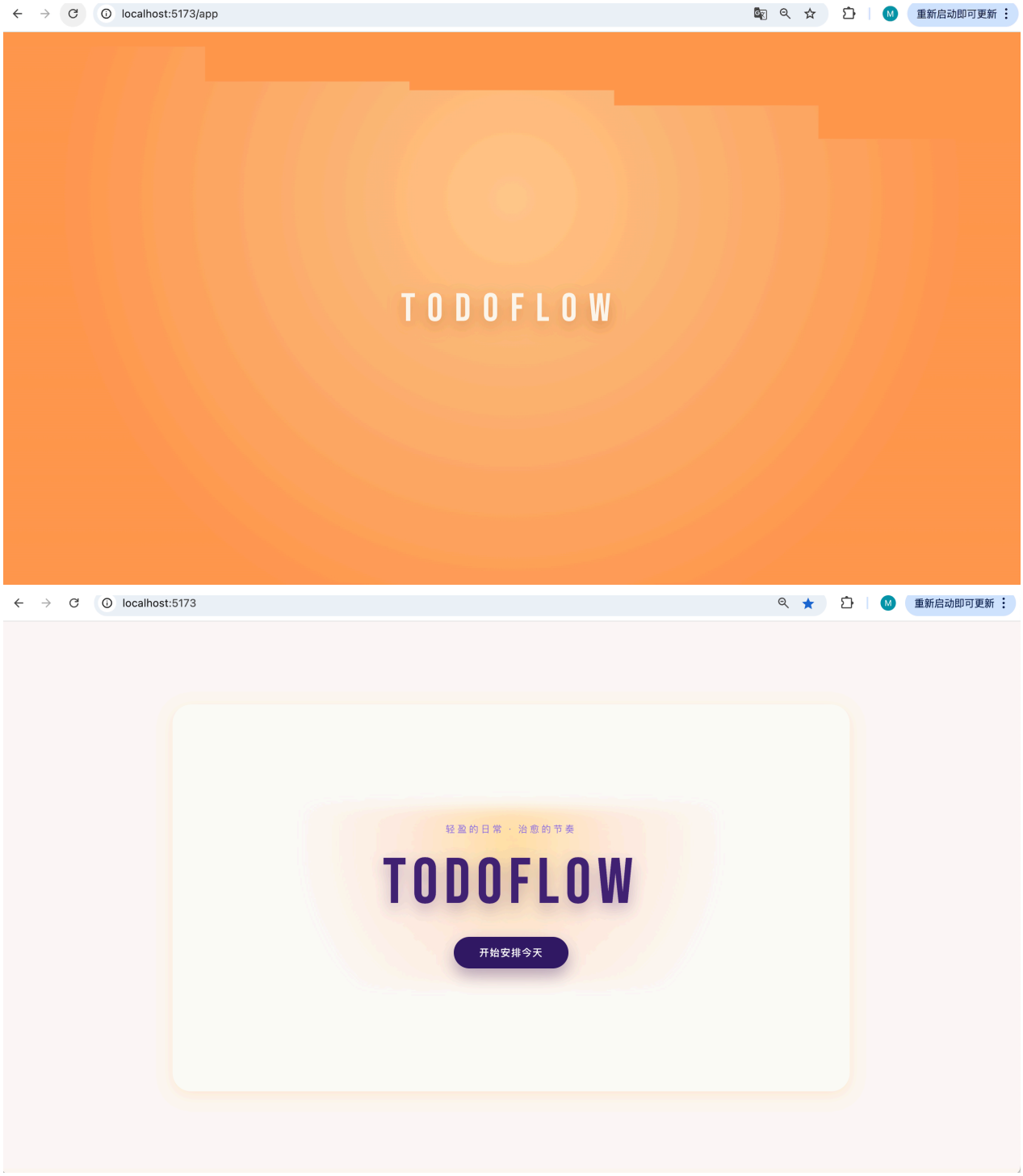
- 已测试过的环境
    - macOS / Safari / Chrome
    - Node.js 20
    - Vite最新版
- 已知问题与不足：
    1. 截止日期未做时区处理 ，截止日期的解析依赖浏览器本地时区。
    2. 同优先级任务的排序策略不够完善。当前系统支持按照 优先级对任务进行排序时，内部通过 `priorityToScore` 将优先级映射为数值，并在排序函数中进行比较：

        ```
        (priorityToScore[b.priority] || 0) - (priorityToScore[a.priority] || 0)
        ```

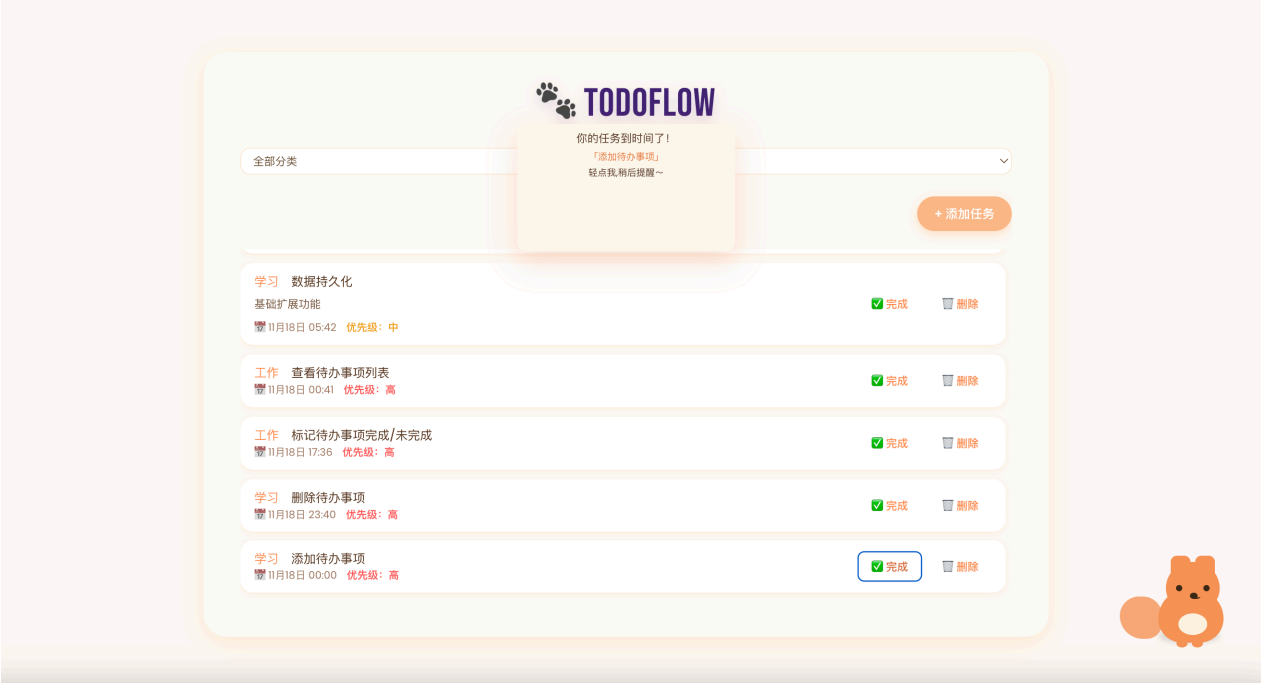
        在同一优先级的情况下，并未进一步定义任务的展示顺序。

6. 总结与反思

- 如果有更多时间，你会如何改进?
    1. 采 UTC 时间统一存储 + 本地时区展示的方案，对截止时间做统一规范化处理，以确保多设备、多时区下的行为一致性。
    2. 同优先级任务按截止时间从近到远排序；如无截止时间，则按创建时间从新到旧排序。
    3. 添加任务统计图表（周完成度、分类占比）
    4. 多设备同步（使用 Firebase 或 Supabase）
- 你觉得这个实现的最大亮点是什么?
    1. 对于提醒功能，使用贪睡机制，贴近真实提醒体验。
    2. 加入欢迎页设计，并且加载时启动过渡动画，提升用户体验。当用户首次打开应用时，会看到一片橙色主题的全屏 Loading Overlay，其上居中显示项目标识 **“TODOFLOW”**。整个启动画面采用柔和的径向渐变背景，并辅以 5 条纵向分片（shards）的动画。分片通过 `scaleY` 动画依次缩小至 0，形成“画面从上往下裂开、逐片揭幕”的视觉效果，使应用主体内容逐步显露。



3. 纯 CSS 实现的提醒助手（Idle + Alert 状态机）



4. 贪睡机制贴近真实提醒体验