这套提示词已经过极度精细的打磨，完全融合了“单 EXE 多版本管理”、“复制自身作为版本卸载器”、“PowerShell 快捷方式”以及“高度定制化的红色毛玻璃 UI”等所有高级需求。

建议你在使用时（如 ChatGPT, Claude, 或 Cursor），将这 7 个提示词按顺序逐个发送，确保每一步的代码逻辑严密。

全局背景设定（请先发给 AI）

> “我要用 Tauri Rust + React + TypeScript + TailwindCSS + FramerMotion 开发一个**单文件多版本管理器**。这是一个高度个性化的红色主题工具。 **核心机制**：
>
> 1. 用户下载一个 `Manager.exe`，里面包含云端多版本下载配置（通过向 exe 末尾追加带有 `<<<CONFIG_START>>>` 标记的 JSON 实现，无需重编译）。
> 2. 运行 `Manager.exe`，UI 展示云端可用版本和本地已安装版本对比，用户可一键安装/重装/卸载。
> 3. **目录隔离**：安装版本 6 时，解压到 `{root_path}\{software_name}\6\`。同时，将 `Manager.exe` 自身复制一份到该目录下，重命名为 `Uninstaller.exe`。
> 4. **系统集成**：
>   - 控制面板：为版本 6 单独写一条注册表 `Uninstall\{software_name}_6`，指向 `...\6\Uninstaller.exe --uninstall --version 6`。
>   - 快捷方式：由 Rust 调用 PowerShell 创建桌面和开始菜单的 `.lnk`，名称为 `软件名 6`。
> 5. **卸载逻辑**：用户在控制面板卸载，系统带 `--uninstall --version 6` 启动对应目录的 `Uninstaller.exe`，仅删除版本 6 文件夹及相关快捷方式/注册表，完成后执行 bat 脚本自删。 请确认理解，接下来我会发送具体的分阶段提示词。”

提示词 1：Rust 核心逻辑 - 目录管理与 PowerShell 快捷方式

> “请帮我编写 Tauri Rust 后端的目录与系统快捷方式模块。
>
> 1. `install_self_as_uninstaller(root_path, app_name, version)`：获取当前运行的 Tauri exe 路径 (`std::env::current_exe()`)，将其复制到 `{root_path}/{app_name}/{version}/` 目录下，命名为 `Uninstaller.exe`。如果文件已存在则覆盖。
> 2. `create_shortcuts_via_powershell(app_name, version, target_exe, icon_path)`：**放弃使用 Rust COM API**。请用 `std::process::Command` 调用 `powershell` 执行脚本生成桌面和开始菜单快捷方式。
>   - 快捷方式名称需拼接版本号：`{app_name} {version}.lnk`。
>   - PowerShell 脚本逻辑：获取桌面和 Programs 目录路径，`$ws = New-Object -ComObject WScript.Shell; $sc = $ws.CreateShortcut('路径'); $sc.TargetPath = '目标'; $sc.IconLocation = '图标'; $sc.Save()`
>   - 请确保 Rust 执行时隐藏 PowerShell 黑框 (`CREATE_NO_WINDOW`)。 请给出完整的 Rust 函数实现。”

提示词 2：Rust 核心逻辑 - 下载、解压与注册表

“请帮我编写 Tauri Rust 后端的下载、解压与注册表逻辑。依赖：`reqwest`, `zip`, `md5`, `winreg`。

1. `start_install(download_url, md5_str, root_path, app_name, version, temp_dir)`：
   - 目标安装目录为 `{root_path}/{app_name}/{version}/`，不存在则递归创建。
   - 在 `temp_dir` 实现带断点续传的下载（利用 `Range` header），计算已下载字节并通过 `app.emit("install_progress", {stage: "downloading", percent})` 推送进度。
   - 计算本地文件 MD5 与 `md5_str` 校验，失败则删除并推送 `{stage: "error"}`。
   - 使用 `zip` crate 解压到目标目录，推送 `{stage: "extracting", percent}`。
   - 解压成功后，调用 `install_self_as_uninstaller` 放置卸载器，调用 `create_shortcuts_via_powershell` 创建快捷方式。
2. `register_uninstaller(app_name, version, root_path)`：
   - 在注册表 `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{app_name}_{version}` 下写入键值。
   - `UninstallString` 设为 `"{root_path}\{app_name}\{version}\Uninstaller.exe" --uninstall --version {version}`。
   - `DisplayIcon` 和 `DisplayName` (如 `软件名 6`) 也需写入。 请给出完整的 Rust 函数实现，注意错误处理。”

提示词 3：Rust 核心逻辑 - 卸载模式与自删除

“请帮我编写 Tauri Rust 的启动控制与卸载清理逻辑。

1. **命令行解析**：启动时读取 `std::env::args()`。如果包含 `--uninstall`，进入卸载模式；同时获取 `--version` 后面的值。暴露 `get_app_mode` 命令返回 `{ mode: "install"/"uninstall", version: Option<String> }`。
2. `perform_uninstall(root_path, app_name, version)`：
   - 推送 `uninstall_progress` 进度。
   - 仅删除特定版本目录：`{root_path}/{app_name}/{version}/`。
   - 调用 PowerShell 删除名为 `{app_name} {version}.lnk` 的桌面和开始菜单快捷方式。
   - 删除注册表项 `HKEY_CURRENT_USER\...\Uninstall\{app_name}_{version}`。
3. **自身删除**：卸载完成后，生成一个 `.bat` 脚本放在系统临时目录，内容为循环 `del` 当前运行中的 `Uninstaller.exe` 并自删。Rust 隐藏窗口启动该 bat 后调用 `app.exit(0)`。 请给出完整的代码实现。”

提示词 4：Rust 核心逻辑 - EOF 配置读取与本地版本扫描

“请帮我编写 Tauri Rust 的配置读取与本地版本扫描模块。

1. `get_config()`：读取当前运行的 `.exe` 二进制流（`std::fs::read`），搜索字符串 `<<<CONFIG_START>>>`。若找到，解析其后所有内容为 JSON（格式 `{software_name, icon_url, versions: [{v, url, md5, changelog}]}`）；若无，返回内置的测试用 JSON。
2. `get_installed_versions(root_path, app_name) -> Vec<String>`：扫描 `{root_path}/{app_name}/` 目录下的所有子文件夹，如果子文件夹下存在 `Uninstaller.exe`，则认为该版本已安装。返回已安装版本号数组。如果目录不存在，返回空数组。 请给出完整的 Rust 函数实现。”

提示词 5：前端 UI 精细打磨 - 主页面 安装与管理模式

“请帮我精细打磨 Tauri 安装器的主页面 UI

React + TailwindCSS + FramerMotion。 **全局风格**：深红背景 (`bg-red-950`)，背景有慢速移动的半透明径向光晕。卡片使用毛玻璃效果 (`backdrop-blur-md`, `bg-white/5`)，强调色 `#ef4444`。无边框窗口，右上角自定义关闭按钮，支持 `data-tauri-drag-region` 拖拽。 **主页面布局** 安装模式：

1. **左侧** 1/ 3宽度：显示软件的 Logo 和软件名。底部是全局设置区，包含一个输入框显示当前安装根目录（默认 `C:\Program Files`），旁边一个‘浏览’按钮（调用 Tauri dialog）。
2. **右侧** 2/ 3宽度：版本列表卡片（滚动列表）。
   - 调用 `get_config` 获取云端版本，调用 `get_installed_versions` 获取本地版本进行对比。
   - 每个版本卡片：版本号大字、更新日志小字。
   - **状态对比 UI**：若本地已安装，卡片右侧显示灰色‘已安装’徽章，并提供‘卸载’和‘重装’（红色幽灵按钮）选项；若未安装，显示醒目的红色实心‘安装’按钮。
   - 点击操作时，底部固定区域弹出进度条（监听 `install_progress`），100% 时变绿并显示打勾动画。 请给出完整的组件代码，注重 Framer Motion 的微交互体验。”

提示词 6：前端 UI 精细打磨 - 卸载页面 由控制面板触发

“请帮我精细打磨 Tauri 安装器的卸载模式 UI

React + TailwindCSS + FramerMotion。 **场景**：用户在控制面板点击卸载，系统带 `--uninstall --version 6` 启动。前端调用 `get_app_mode` 得知当前为卸载模式且版本为 6。 **卸载模式布局**：

1. 极简居中布局，窗口尺寸 500x350。
2. 顶部显示一个红色的警告图标（Framer Motion 做轻微的呼吸缩放动画）。
3. 中间大字提示：`准备卸载 [软件名] [版本号]`，下方小字提示：`此操作将删除该版本的所有文件和快捷方式，且不可恢复。`
4. 底部并排两个按钮：左侧灰色幽灵按钮‘取消’（点击 `app.exit(0)`），右侧红色实心按钮‘立即卸载’。

5. 点击‘立即卸载’后，调用 `perform_uninstall`，按钮变不可点击状态，下方出现极简的红色细线进度条（监听 `uninstall_progress`）。完成后显示‘卸载成功，窗口将自动关闭’，2 秒后 `app.exit(0)`。 请给出完整的组件代码。”

提示词 7：Node.js 安装包生成器

“我需要一个 Node.js 编写的‘安装包生成器’脚本 (`generator.js`)。 **前提**：同目录下有 `base_installer.exe` Tauri编译好的基础包, `rcedit.exe`, `config.json` 包含$software_name, icon_url, versions$数组。 **需求**：

1. 读取 `config.json`。
2. 如果 `icon_url` 是网络链接，使用 `https` 模块下载保存为本地 `temp.ico`；如果是本地路径则直接使用。
3. 将 `config.json` 内容转为字符串，前面拼接标记 `<<<CONFIG_START>>>`。
4. 读取 `base_installer.exe` 为 Buffer，将标记和 JSON 字符串追加到 Buffer 末尾，写入 `final_installer.exe`。
5. 使用 `child_process.execSync` 调用 `rcedit.exe final_installer.exe --set-icon temp.ico` 替换图标。
6. 删除 `temp.ico`，控制台输出‘生成成功’。 请给出完整的 Node.js 脚本代码。”