

案例研究：文档驱动·人机协作：AI Agent 开发方法论实证

项目概述

0. 文档索引

1.1 个人品牌与网站愿景文档

1.2 功能清单 (MVP + 扩展)

2.1 网站结构文档 (Sitemap)

2.2 内容规划表

2D portfolio AI辅助指南

3.1 风格指南

3.2 原型与线框图

3.3 交互与动画说明

4.1 技术栈与架构文档

4.2 项目文件结构文档

4.3 视觉资产与使用规范

5.1 编码规范文档

5.2 开发任务分解表

6.1 部署说明文档

6.2 部署环境配置表

7.1 内容更新指南

7.2 SEO 与访问性说明

资源详情

更新记录

完整技术规范书与开发计划

VN游戏感改造执行第2季手册

VN游戏感改造执行手册

完整技术规范书与开发计划

层面	选择	版本/备注
框架	Next.js 16	App Router, React 19
语言	TypeScript Strict	tsconfig.json strict: true, 全量 TS
包管理	pnpm	最新稳定版
CSS	Tailwind CSS v4	自定义设计令牌
UI 组件	shadcn/ui	基础骨架 + 覆写样式
动画	Framer Motion	唯一动画库, 无 GSAP
字体	全自托管 + next/font	JP: M PLUS Rounded 1c, EN: Nunito, ZH: Noto Sans SC
数据校验	Zod	校验 case-studies.json / vn-scenario.json
i18n	next-intl	path-based 路由策略 (/[lang]/...)
数据层	本地 JSON + messages JSON	无 CMS, 无数据库; Case Study 正文结构化直存于 messages JSON
Markdown 渲染	不使用	Case Study 正文统一为 messages JSON 结构化字段, 通过 TypeScript + next-intl 读取
CI	GitHub Actions	lint + typecheck + build
部署	Vercel	auto deploy from main, PR previews
联系表单	纯 UI 展示	无后端逻辑
测试	无	v1 不引入测试框架, 依赖 TypeScript + ESLint + Build 保障
分析监控	无	v1 不接入
原型方式	代码驱动设计	辅以结构化设计文档 (自然语言 + 小标题 MD 文件)
依赖更新	手动	不启用 Dependabot / Renovate

二、设计系统

2.1 设计令牌 (Design Tokens)

--color-primary: #FFB7C5;

--color-primary-soft: #FFEBEC;

25 backlinks 4,435 words 21,597 characters

OC | OpenCode.ai/docs access: X

[] Update directory structure based on official docs

[] Add/update rules section

[] Add/update themes section

[] Add/update commands section

[] Add skills section

[] Add/update models section

[] Add/update agents section (expand beyond AGENTS.md)

[] Add GitHub integration section

[] Add plugin development section

[] Add/update providers directory section

[] Update loading priority list

[] Update AGENTS.md section to clarify behavior guidelines vs agent system

[] Output summary table of all changes

+ Thought: 408ms

% WebFetch https://opencode.ai/docs/rules/

% WebFetch https://opencode.ai/docs/themes/

% WebFetch https://opencode.ai/docs/commands/

% WebFetch https://opencode.ai/docs/skills/

% WebFetch https://opencode.ai/docs/models/

% WebFetch https://opencode.ai/docs/agents/

% WebFetch https://opencode.ai/docs/github/

% WebFetch https://opencode.ai/docs/providers/

% WebFetch https://opencode.ai/docs/plugins/

% WebFetch https://opencode.ai/docs/config/

+ Thought: 10.7s

Todos

[✓] Fetch all relevant official docs pages (rules, themes, commands, skills, models, agents, github, providers, plugins)

[•] Write comprehensive updated file with all new sections

[] Output summary table of all changes

+ Thought: 327ms

~ Preparing write...

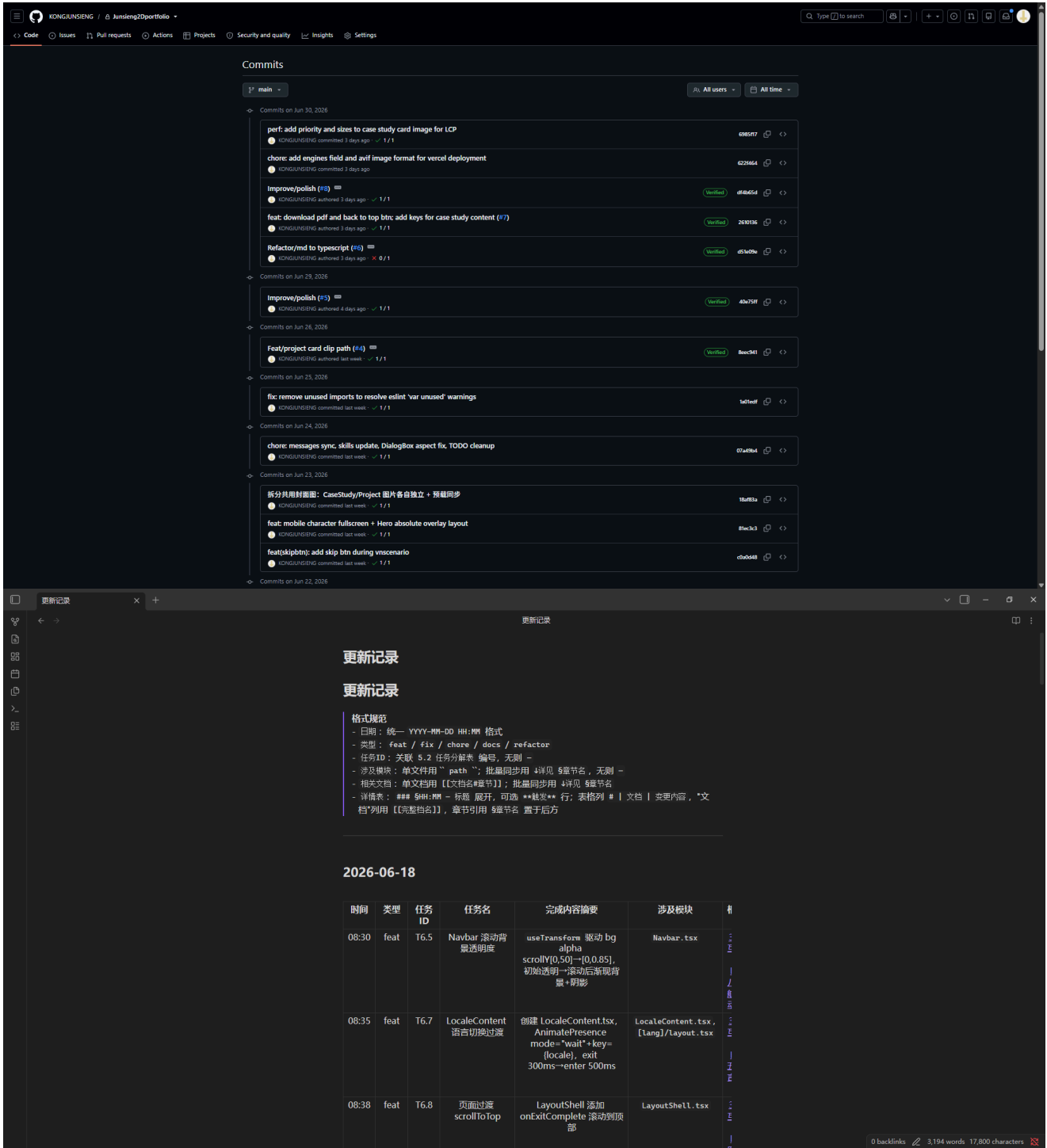
■ Build · DeepSeek V4 Flash Free

Build · DeepSeek V4 Flash Free OpenCode Zen

esc interrupt

73.8K (37%) ctrl+p commands

OpenCode 1.15.13



本项目是求职作品集的一部分，目标受众为科技公司的 HR、技术管理者与开发者同行。最终交付成果为 JunsienGPortfolio 展示型个人作品集网站（即本网站）。项目在管理与技术上的独特之处在于：

- 将软件工程标准的文档体系与工作流引入个人项目，实现全流程可追溯
- AI Agent 在文档驱动下生成代码，而人力专注于质量管理、测试、翻译及 Bug 修复，与 Agent 形成协作闭环。

背景与动机

选择个人作品集网站作为实验场景，原因有二：

- 想做

- 规模适中且具备完整的项目生命周期（从需求到部署）

采用 **文档先行** 方法的核心原因有二：

- 避免 AI Agent 生成不符合预期的代码，通过文档建立明确的约束与指引
- 使整个流程可复现，为后续项目提供标准化参考

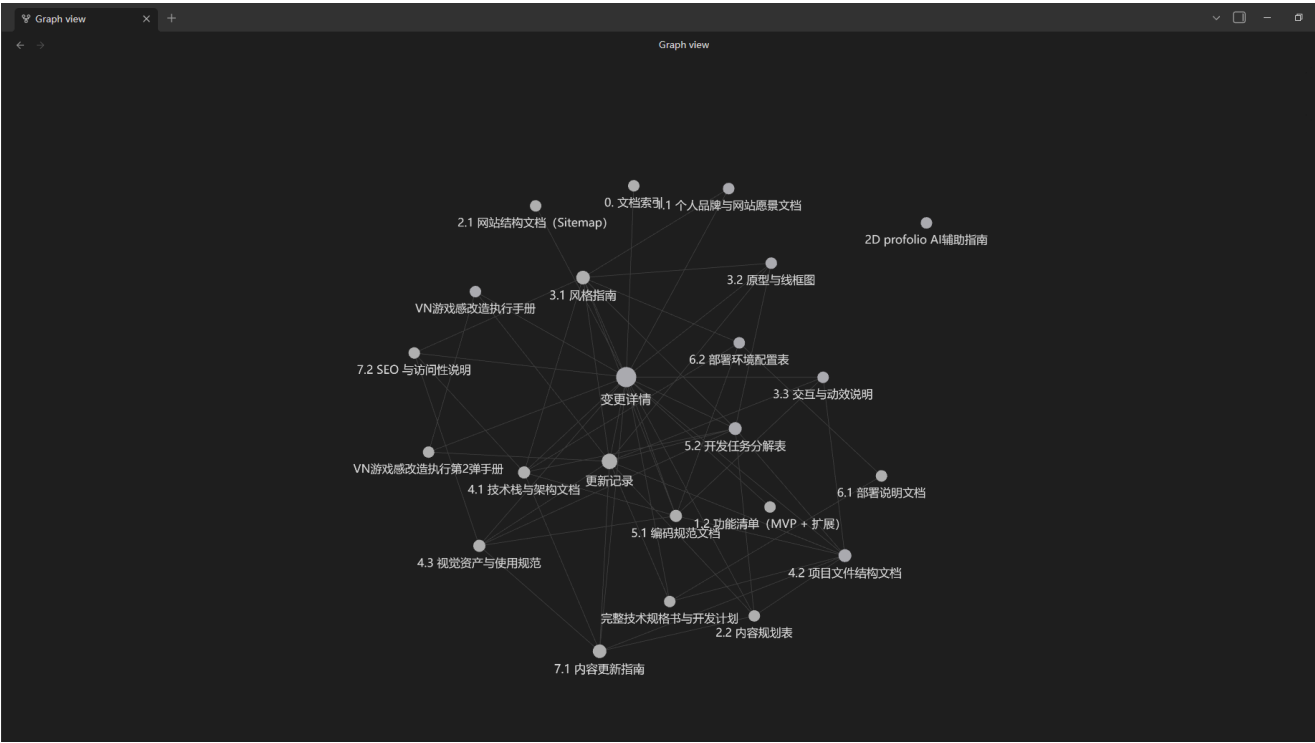
选择 **OpenCode Agent** 则是为了探索新技术路线，验证文档驱动开发在实际项目中的可行性。在工具选型上：

- **Obsidian**：采用 Markdown 格式，既便于 AI Agent 解析处理，也保障了人的可读性
- **Git**：承担文档与项目代码的统一管理与版本控制
- **Vercel**：作为免费部署方案的首选

方法论与架构

文档先行的理念与流程

按照软件工程标准建立编号 **1.1–7.2** 的文档体系，覆盖从品牌愿景到运维运营的完整生命周期。文档关系图明确了依赖链路。



涵盖以下核心文档：

编号	文档名称	说明
1.1	个人品牌与网站愿景文档	定位、目标、品牌调性
1.2	功能清单（MVP + 扩展）	功能范围与优先级
2.1	网站结构文档（Sitemap）	页面结构与路由设计
2.2	内容规划表	页面内容与 i18n 键值规划

编号	文档名称	说明
3.1	风格指南	设计令牌与 UI 规范
4.1	技术栈与架构文档	技术选型与架构决策
4.2	项目文件结构文档	目录结构与组件职责
4.3	视觉资产与使用规范	图片、字体资源规范
5.1	编码规范文档	命名、TS 规则、约定
5.2	开发任务分解表	105 个任务分解与排期
6.1	部署说明文档	Vercel Runbook 与流程
6.2	环境配置表	Node.js、pnpm 版本锁定
7.1	内容更新指南	持续维护操作流程
7.2	SEO 与无障碍说明	SEO 元数据与 a11y 规范

(完整清单见附录 A)。另设 VN 游戏感改造执行手册系列作为专题分支。全程使用 Obsidian + Git 管理，确保版本可追溯。

Agent 配置与文档交互

`opencode.json` 配置了 `external_directory` 权限，允许 Agent 读取本地 Obsidian 文档库。 `AGENTS.md` 注入项目上下文。在每个开发会话中，手动引用对应文档指引 Agent 理解需求、架构和任务约束，Agent 以文档为唯一事实源生成代码。

系统架构

层面	选型	要点
框架	Next.js 16 App Router + React 19	全 TypeScript Strict
CSS	Tailwind CSS v4	自定义设计令牌 (<code>@theme</code>)，无 <code>tailwind.config</code>
动画	Framer Motion	唯一动画库，含游戏风 variants
i18n	next-intl	path-based (<code>/[lang]/...</code>)，三语 (zh/ja/en)
数据层	本地 JSON + Zod 校验	零 CMS/数据库， <code>read-data.ts</code> 集中读取
字体	全自托管	JP/EN: <code>next/font/local</code> ，ZH: <code>@font-face</code> 13 子集
UI	自定义组件	无第三方 UI 库；游戏风组件
包管理	pnpm	CI 使用 <code>--frozen-lockfile</code>

组件架构遵循 Server/Client Component 边界：根布局为 Server Component，LayoutShell 和 LocaleContent 为 Client Component 处理动画与交互状态。（完整版本明细见附录 C）

部署方案

维度	说明
部署平台	Vercel Hobby（免费），域名 jungsieng-portfolio.vercel.app
CI 工具	GitHub Actions，Node.js 22.x，流程：lint → typecheck → build
自动触发	main 推送 → Production；PR 创建/更新 → Preview
环境变量	v1 无需配置
回滚方式	Vercel Dashboard → Deployments → Promote to Production
详细配置	见附录 D

实施过程

开发周期 13 天（2026-06-16 至 2026-06-30，每天约2-3小时），按文档驱动、阶段推进的方式划分为 7 个阶段。每个阶段在 **OpenCode Agent** 读取对应设计文档后启动开发，Agent 生成代码初稿，人力进行审阅、修正、验证与文档同步。

阶段一：项目骨架搭建与数据层建立（Day 1）



初始化 Next.js 16 App Router + TypeScript Strict + Tailwind v4 项目骨架，安装核心依赖，搭建目录结构与自定义设计令牌。创建 Zod schema 约束数据文件，同步建立数据文件，配置国际化路由与统一数据访问层。搭建 CI 流水线。

阶段二：全局布局与首页核心区块（Day 2）

全局布局层创建导航栏、页脚与语言切换器。首页五大 Section 同步开发——HeroSection（立绘+文字双列，stagger 入场序列）、CaseStudiesSection、ProjectsSection、AboutSection、ContactSection。Case Study 详情页实现动态路由与服务端 SEO。阶段末期修复集成问题并通过全验证。

阶段三：动效深化与 VN 游戏感 UI 系统（Day 3）

创建 4 组 VN 风格组件并集成至各 Section。同步完成 Hero 视差、导航栏滚动渐变与语言切换过渡等动效增强。

阶段四：Intro 序幕系统开发（Day 4–Day 6）

开发完整的游戏风格序幕系统——四段式状态机（开场动画 → 标题画面 → 情景播放 → 加载过渡），支持对话推进、选项分支、角色立绘切换与鼠标排斥、自动推进等交互。三语剧本共 43 场景，含多分支与 6 个终场。

阶段五：移动端适配与交互增强（Day 7–Day 10）

移动端布局与交互重构。新增卡片展开详情与空闲动画，同步完成项目内容充实与三语同步。

阶段六：代码清理与文档体系重构（Day 11–Day 12）

系统性移除未使用代码与导出。将更新记录拆分为摘要表与详细变更追踪的双层文档结构，同步修订开发任务分解表。

阶段七：数据层重构与部署准备（Day 13）

Case Study 详情正文从 Markdown 编译链路迁移至 messages JSON 直接读取，重构模板渲染结构。新增下载按钮与回顶按钮，配置图片格式与引擎约束。代码全验证后推送至 main 分支，准备 Vercel 部署。

各项指标详见下文成果表格。每次 Agent 行为与文档预期偏离时，均通过“代码修正 → 文档同步”的双向反馈机制进行校正，形成 **人机协作** 的闭环迭代模式。

关键挑战与解决方案

挑战一：OpenCode 的复杂配置

描述： 从零接触 OpenCode Agent 时，面临陡峭的配置学习曲线。基础功能之外，进阶配置涉及 permission 权限粒度（external_directory 允许 Agent 读取外部 Obsidian 文档库）、skill 引用注入、AGENTS.md 项目上下文定义、token 管理等多个维度。官方文档虽详尽但结构庞大，实际操作中配置顺序、参数之间的相互影响、各项配置的必要性判断，均需大量试错才能掌握。

应对策略： 避免完全依赖 AI 聊天工具或死磕官方文档。AI 聊天工具（如 ChatGPT 等）提供的配置指引常有过时或不准确之处，将其定位为「进阶搜索引擎」以发现资料来源。具体学习

路径为：先在 GitHub 等平台参考他人的配置实践建立初步理解，再结合聊天工具与官方文档进行验证操作，最终在反复实践中形成对自身配置体系的完整认知。

挑战二：需求讨论与文档编写的时间投入

描述： 文档先行法的固有前期成本。在代码开发之前，独立投入约 8 小时 25 分钟完成 20+ 份编号文档的编写，涵盖从品牌愿景到运维运营的完整生命周期。每一份文档均经过详细的需求讨论、解决方案论证与逐项填写，迭代较少但耗时显著。

应对策略： 这是一项需根据项目性质权衡的设计决策。文档粒度直接决定 AI Agent 输出与预期的偏离程度——本项目中用户对需求有清晰认知且追求高精度，因此选择投入充分时间编写详尽文档。若项目目标更倾向于快速原型验证、由 AI 主导创意发散，则可在文档环节适当缩减投入，接受一定程度的偏差在后续迭代中修正。

挑战三：AI Agent 难以主动提点知识盲点

描述： AI Agent 本质上被设计为顺应使用者的指令，多数情况下仅跟随后者的逻辑推进，不会主动识别或指出使用者认知中的盲区。当使用者对某一领域缺乏经验时，其逻辑本身可能包含不标准的假设或遗漏关键考量。例如项目中 Zod schema 遗漏 `characterImage` 字段导致立绘无法切换——使用者并未意识到需要声明该字段，Agent 也未主动提示。

应对策略： 此问题在当前 AI 模型能力与 prompt 精度下无完美解法。有效的缓解手段是通过特定的 prompt 设计引导 Agent 转换角色，例如要求其「以行业标准批判现有方案」或「列举本领域常见但尚未被考虑的技术风险」。Agent 的智能程度、使用者的逻辑清晰度与 prompt 的准确度三者共同决定该挑战的影响程度。

挑战四：人类难以精确描述 UI 感受

描述： UI 微调场景下，间距、色调、排版等主观感知层面的效果难以量化为 AI Agent 可执行的精确指令。Agent 只能理解具体数值（`px`、`rem`、色值编码），而人类对界面的感受是整体性的——「这个按钮看起来不够突出」「这两行文字的间距感觉不对」等描述对 Agent 而言缺乏可操作性。

应对策略： 依据是否有参照物采取二分策略。当有参考对象时，直接给出示例（「与 `SectionTitle` 组件的间距相同」「参考某网站的按钮样式」）帮助 Agent 建立参照系。当无参照物且涉及精细调整时，由开发者直接阅读并修改代码，效率与效果均优于反复通过自然语言描述视觉感受让 Agent 推测意图。

挑战五：文档与实际实现的偏差

描述： 设计文档中的规格与实际代码实现之间存在偏差。偏差的根源主要是需求的自然演进——开发过程中产生了更优的解法或新的想法，导致代码脱离原始文档的规格。例如 DialogBox 组件经历 CSS 方案→PNG 素材→CSS 三层圆角结构的三次方案演进，每次迭代均在代码先行实现后再回头同步文档。

应对策略： 偏差不可避免，应对的关键在于建立可靠的同步机制。个人选择「先改代码、再同步文档」的工作流：优先在代码中验证新方案的可行性，确认后再将变更反映到对应的设计文档中。这属于个人偏好而非行业标准，也可选择「先改文档、再改代码」的流程以确保文档始终是先导。重要的是保持两者最终的一致性，而非追求偏差的绝对避免。

挑战六：文档同步的高维护成本

描述： 每次代码变更后需将对应设计文档与实际实现对齐维护。本项目中单次批量同步涉及 6 至 22 份文档不等，需逐份检查字段描述、组件接口、数据结构等是否与代码一致。项目后期更将 462 行的更新记录拆分为摘要表与详细变更追踪的双层结构以降低维护负担。

应对策略： 维护工作本身通过 AI Agent 的 prompt 驱动完成，耗时可控。关键挑战在于人工需记录准确的变更上下文以供 Agent 填入。未选择将文档同步需求写入 AGENTS.md 使其自动化，原因有三：

- AGENTS.md 指令的稳定性不足
- 占据上下文窗口可能影响核心任务
- 自动化可能使 Agent 在关键功能开发中失去焦点

不减文档粒度——文档既是未来维护的关键参考，也是工作过程的完整证明。

挑战七：AI Agent 在长会话中失去 Focus

描述： 随会话进行，上下文窗口逐步累积，AI Agent 对当前任务目标的注意力逐渐分散，导致工作质量下降。Agent 可能遗忘早期约定的约束条件、混淆已修改的文件状态、或在复杂任务中偏离核心需求。

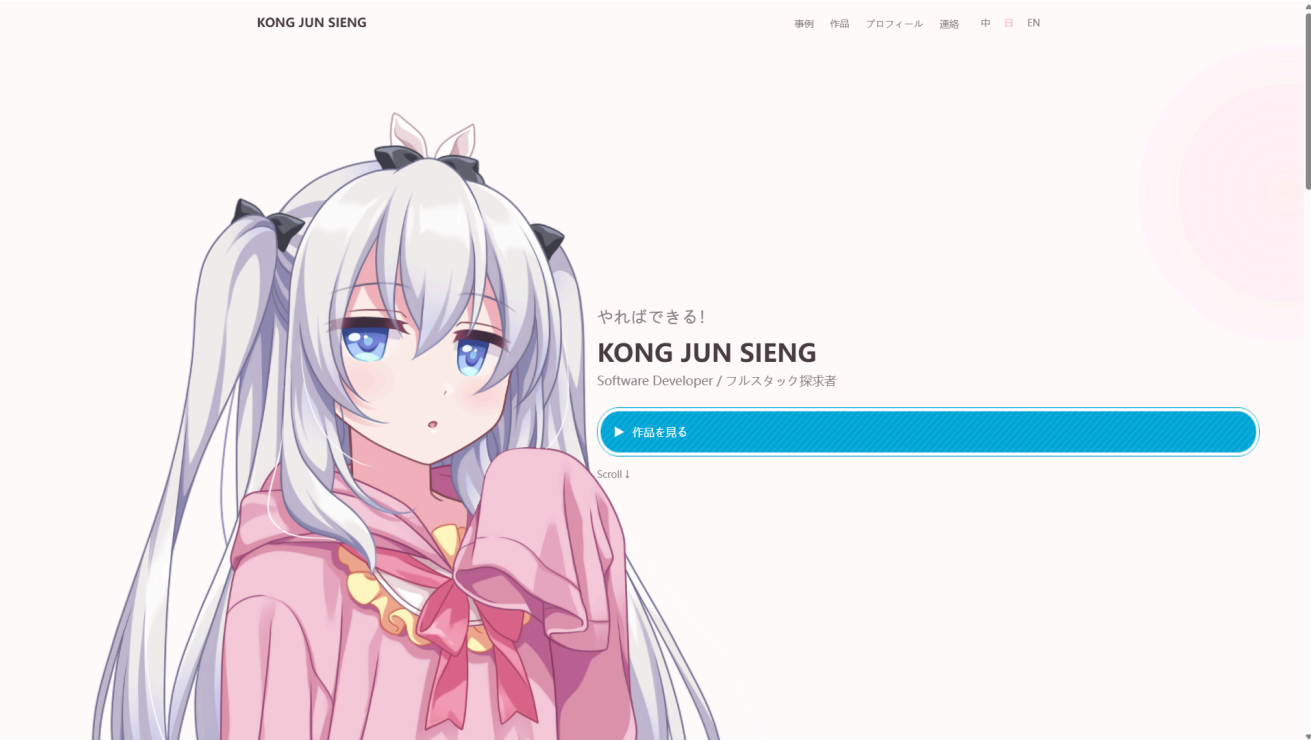
应对策略： 一个会话专注一种功能，完成后关闭会话重新开启。**OpenCode Agent** 配置本地 Ollama 模型运行，不计 token 消耗，频繁重开会话不会产生额外成本。此策略对按 token 计费的付费 Agent 产品不够经济，但后者在社群中已有其他解决方案（如会话摘要压缩、子代理分工等）。

成果与展示

交付物总览

维度	成果
代码规模	60+ 文件，覆盖组件、数据层、国际化、样式、配置全链路
开发任务	105 个（Phase 1–12），100% 完成
设计文档	20+ 份编号文档（1.1–7.2），含 VN 游戏感改造手册
开发周期	13 天（文档阶段另计约 8 小时 25 分钟）
人机协作比	Agent 负责代码生成与文档同步，人力负责需求定义、质量审核、Bug 修复与部署

功能特性



首页五大 Section：HeroSection（全屏角色立绘 + 视差 + 光晕脉冲动画）、CaseStudiesSection（案例研究卡片 + 空闲微摆动效）、ProjectsSection（双列网格 + clip-path 圆形展开详情）、AboutSection（对话框叙事 + ATS 技能标签）、ContactSection（邮箱/社交/简历下载，纯展示无表单）。

KONG JUN SIENG

SOFTWARE DEVELOPER PORTFOLIO

▶ 設定

v1.0

▶ スキップ

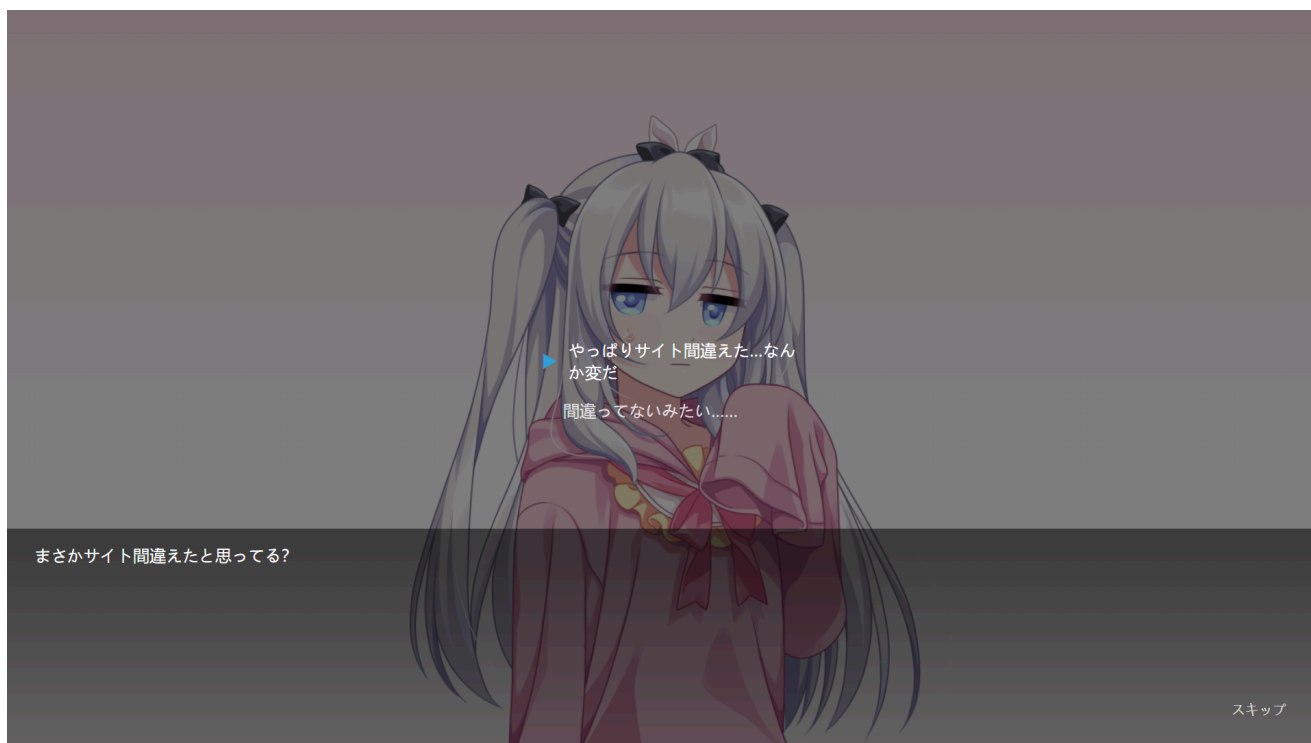


ルカちゃん

まさかサイト間違えたと思ってる？

▶ スキップ

VN 游戏风序幕系统：四段式 Intro 流程（开场动画 → 标题画面 → 情景播放 → 加载过渡），43 场景三语剧本、7 角色表情、多分支选项、鼠标排斥交互、自动推进、Skip 跳过。访客着陆后先体验序幕，再进入首页。



游戏风 UI 系统：章节标题系统（CHAPTER 标签 + 蓝色下划线）、蓝色对话框叙事、选项按钮交互、过渡台词、动态导航栏滚动效果、Framer Motion 全套 variants（staggerContainer / chapterReveal / dialogSlideUp 等）。

三语国际化：zh/ja/en 完整三语支持，path-based 路由（/[lang]/...），next-intl 驱动。

Case Study 详情页：9 节结构化正文（executiveSummary → appendix），PDF 下载，BackToTop 回顶按钮，服务端 generateMetadata SEO。

代码与架构质量

- **全 TypeScript Strict：** tsconfig.json 启用 strict + noUncheckedIndexedAccess + noImplicitReturns，所有组件 Props 使用 Readonly<{...}> 包装
- **Tailwind CSS v4 设计令牌：** @theme 定义完整色彩系统（primary/secondary/accent/dialog-blue 等）、字体、间距、圆角、阴影
- **Zod 数据校验：** 所有 JSON 数据文件通过 schema 类型检查，零 CMS/数据库
- **CI/CD：** GitHub Actions 执行 lint → typecheck → build，Node.js 22.x
- **Server/Client Component 边界：** 严格分离服务端数据读取与客户端动画交互

线上链接

生产环境：<https://jungsieng-portfolio.vercel.app>

讨论

文档先行 + Agent 模式的优势分析

文档先行法的前期投入回报比相当显著——首轮 Agent 生成即覆盖了大部分功能骨架。项目后期新增需求较原始文档增加约 40% 且复杂度提升，这是开发周期延伸的主因——说明文档先行模式在初始阶段的时间优势突出，项目延期的根源在于需求的自然演化，而非流程本身。

Agent 在文档约束下的输出稳定性是另一核心收益。明确的架构约定、组件接口定义和设计令牌规范，使 Agent 生成的代码在首次提交时即基本符合项目标准，减少了反复澄清需求、对齐约定的沟通成本。

OpenCode CLI 的天然优势

OpenCode 的命令行界面设计在开发流程中带来了两项独特优势：

- **多窗口并行开发：** OpenCode CLI 的轻量化架构允许同时开启多个独立会话窗口，各自专注不同功能模块（如一个窗口开发组件、另一个窗口同步文档），互不干扰。这有效缩短了串行等待时间，但前提是每个窗口的任务必须有明确的文档约束与质量验收标准——若缺乏文档基准，多窗口并行反而可能导致代码风格割裂或架构不一致。
- **本地部署保障数据安全：** OpenCode 原生支持接入本地 Ollama 模型，为涉及机密资料或未公开业务逻辑的项目提供了理想方案。Agent 开发所需的全部代码与文档可完全在本地运行，无需上传至任何第三方 API 服务。

固有局限

该模式在 UI 微观调整环节最吃力。间距、色调、排版等主观感知层面的效果难以量化为 Agent 可执行的具体指令。Agent 只能理解数值（px、rem、色值编码），而人类对界面的感受是整体性的。描述与讨论的时间投入与最终产出满意度在 UI 微调场景中不成正比。当无参照物且涉及精细调整时，由开发者直接修改代码，效率与效果均优于反复通过自然语言描述视觉感受让 Agent 推测意图。

流程与角色重塑

该模式将开发者的角色从“编写代码”迁移至“定义需求、做出决策、审查代码”。开发者不再逐行编写实现，而是聚焦于将需求转化为精确的文档规格、在关键节点做出技术决策、验证 Agent 生成代码的质量与一致性。

这一转变对基本功与行业经验提出了更高要求。AI Agent 本质上被设计为顺应使用者的指令，多数情况下仅跟随后者的逻辑推进，不会主动识别或指出使用者认知中的盲区。当使用者对某一领域缺乏经验时，其逻辑本身可能包含偏离标准的假设或遗漏关键考量，Agent 难以自动弥补。一个有效的缓解手段是通过 prompt 设计引导 Agent 转换角色（例如要求其“以行业标准批判现有方案”），但最终方案质量仍高度依赖使用者的逻辑清晰度与领域认知深度。

团队场景推演

若将此模式扩展至团队场景（每位成员配有 Agent），预期会呈现分化特征。前期文档讨论阶段因涉及多角色协调（UI 设计、前端、后端、测试），沟通成本将显著增加，需要 Tech Lead 先行规定整体框架，再由各角色细化各自领域文档。开发与测试阶段因文档驱动而具备快速初始化优势，效率与质量取决于各角色文档的详细程度与准确性。需要指出的是，此推演基于个人项目经验，未经商业团队全流程验证。

HR/管理层视角

候选人主动尝试 AI Agent 开发方式并建立完整流程，本身是加分项。一个可能的问题是“既然有 Agent 辅助，为何开发周期仍有相当长度”——原因有二：

- Agent 的配置与调优存在学习曲线，效果取决于使用者对工具的掌握程度
- 文档粒度与输出稳定性之间存在直接权衡——文档越详尽（前期投入越大），Agent 输出越稳定；选择高速初始化+大量优化还是较慢初始化+少量优化，取决于项目对交付速度与质量的不同诉求

开源软件与商用边界

所选技术栈（Next.js、React、Tailwind CSS、Framer Motion、Zod 等）均采用宽松许可证（MIT、Apache 2.0 等），个人作品集或商业项目使用均无合规顾虑。**OpenCode Agent** 同为开源工具，企业在将其纳入商业化流程时，需自行审阅许可证条款。

另外，关于“AI Agent 开发者能力”的常见顾虑：有人认为使用 Agent 开发的开发者可能缺乏真实能力。恰恰相反——Agent 的顺应性设计使其高度依赖使用者的行业经验与逻辑判断力。Agent 用得越有效，越能反映使用者在需求拆解、架构设计、质量把控等方面的基本功水平。

反思与最佳实践

文档密度优先于迭代修复

最核心的收获是验证了一条原则：**在文档环节投入充分时间建立高质量基准，其长期回报远高于“快速启动→反复修正”的迭代路径**（具体数据见“关键挑战 → 挑战二”章节）。这与常见的“快速原型验证”倡导形成对比——后者更适合需求模糊或创意发散型项目，而本案例证明当开发者对交付质量有明确预期时，前期文档投入是效率最优解。

若有下一个类似项目，我会在保持同等文档密度的基础上，尝试更多的 Agent 配置变体（如 skill 链式组合、多 Agent 分工等），以探索不同配置策略对输出质量的影响边界。流程设计虽已尽量参照行业软件工程规范，但因个人缺乏商业团队工作经验，规范的完整性与正确性有待在实际团队中进一步验证——这也划定了个人能力边界，指明了未来需要重点学习的领域。

Agent 开发能力的认知升级

从零配置 **OpenCode Agent** 的过程带来了对 AI 开发工具链的体系化理解。skill（专项技能注入）、MCP（Model Context Protocol）、AGENTS.md（项目上下文定义）三者之间的协作关系——skill 提供领域专精知识，AGENTS.md 注入项目级约束，MCP 扩展工具边界——构成了 Agent 配置的核心骨架。

在 prompt 工程方面，“角色转换提示”被验证有效：要求 Agent“以行业标准批判现有方案”或“列举本领域常见但尚未被考虑的技术风险”，能够部分缓解 Agent 顺应性设计带来的盲区问题。但 prompt 的效果高度依赖使用者自身的领域认知深度——使用者必须先识别出可能存在的盲区，才能设计相应的提示来引导 Agent 弥补。AI 辅助开发时代对开发者基本功的要求不是降低，而是提高。

工具选型的务实建议

对于有意尝试 Agent 开发流程的同行，工具选型建议如下：若预算充足且追求最低上手门槛，Claude（Anthropic）作为当前最成熟的 Agent 产品是稳妥选择；若寻求低成本入门且希望保留本地部署能力，**OpenCode** 是优解——其开源特性允许在敏感项目中使用本地 Ollama 模型，无需将代码或文档上传至第三方服务。从 OpenCode 入门 Agent 开发流，能够在零成本环境下建立对 Agent 配置、文档交互、会话管理等核心概念的完整认知，再迁移至其他付费产品时也将更加得心应手。

结论与展望

核心发现

文档先行 + AI Agent 的开发模式在个人项目中具有显著的可操作性与投资回报率。核心机制在于：以软件工程标准的文档体系（20+ 份编号文档覆盖完整生命周期）作为 Agent 的约束框架与事实源，将人的优势（需求定义、架构决策、质量审核）与 Agent 的优势（代码生成、文档同步、重复劳动自动化）形成 **协作闭环**。该模式在首次开发会话中即实现约 80% 功能骨架的一次性高命中率生成，验证了文档密度与 Agent 输出质量之间的正相关关系。

从更宏观的视角来看，AI 辅助开发时代对开发者能力的定义正在重塑：工具门槛降低的同时，对需求拆解能力、架构判断力和质量把控力提出了更高要求。Agent 用得越有效，反而越能彰显使用者在这些基本功维度的水平。

适用场景

该模式最适合以下场景：

- 开发者对项目需求有较为清晰的预期，追求交付质量而非快速原型验证
- 项目规模适中（个人项目或小型团队），文档维护成本可控
- 开发者具备一定的软件工程认知，能够编写结构化的设计文档

对于需求模糊、以创意发散为导向或开发周期极短的项目，适当降低文档粒度以换取启动速度可能是更务实的选择。

未来方向

后续探索有几个方向。首先，计划将 Claude 或其他主流 Agent 产品纳入对比测试，在同一文档体系下评估不同 Agent 的输出质量与开发效率差异。其次，该文档先行 + Agent 驱动的流程将复用到其他个人项目（如游戏开发计划，目前还在构思阶段），验证方法论在跨领域项目中的可迁移性。最后，文档流程自动化——若能通过 Agent 或脚本工具自动生成或同步部分文档内容，将进一步提升整体效率。上述方向的共同目标是：将单次实践逐步演进为可标准化、可复用的个人开发方法论。

关键词

OpenCode Agent，文档先行，Obsidian，Git，Vercel，个人作品集，Next.js，AI 驱动开发，人机协作，游戏风 UI，Ollama

附录

附录 A：文档体系完整清单

编号	文档名称	说明
0	文档索引	全局文档地图与交叉引用
1.1	个人品牌与网站愿景文档	定位、目标、品牌调性
1.2	功能清单（MVP + 扩展）	功能范围与优先级
2.1	网站结构文档（Sitemap）	页面结构与路由设计
2.2	内容规划表	页面内容与 i18n 键值规划
3.1	风格指南	设计令牌与 UI 规范
3.2	原型与线框图	布局与信息层级
3.3	交互与动效说明	动画 spec 与交互行为
4.1	技术栈与架构文档	技术选型与架构决策
4.2	项目文件结构文档	目录结构与组件职责
4.3	视觉资产与使用规范	图片、字体资源规范
5.1	编码规范文档	命名、TS 规则、约定
5.2	开发任务分解表	105 个任务分解与排期
6.1	部署说明文档	Vercel Runbook 与流程
6.2	环境配置表	Node.js、pnpm 版本锁定
7.1	内容更新指南	持续维护操作流程
7.2	SEO 与无障碍说明	SEO 元数据与 a11y 规范
—	VN 游戏感改造执行手册	VN 风格改造规格与实现记录
—	VN 游戏感改造执行第 2 弹手册	Intro 序幕系统规格与实现记录
—	完整技术规格书与开发计划	综合技术规格
—	AI 驱动开发概要与工作区规则	Agent 配置与开发流程
—	更新记录	变更日志摘要（108 行）
—	变更详情	详细变更追踪（394 行）
—	TODO 开发清单	阶段性检查清单

附录 B：OpenCode Agent 配置要点

opencode.json 核心配置：

```
{
  "instructions": [".opencode/skills/frontend-design/SKILL.md"],
  "permission": {
    "external_directory": {
      "E:/Xeno/Obsidian/2DportfolioDocuments/**": "allow"
    }
  }
}
```

```
    },
    "edit": {
      "E:/Xeno/Obsidian/2DportfolioDocuments/**": "ask"
    }
  }
}
```

- `external_directory`：授予 Agent 读取 Obsidian 文档库的权限，使 Agent 能在开发会话中直接访问全部 20+ 份设计文档
- `edit.ask`：对 Obsidian 文档的修改需人工确认，防止 Agent 未经审核即更改设计文档
- `instructions`：加载 `frontend-design skill`，为 Agent 注入前端设计决策指南
- **AGENTS.md**：在项目根目录注入项目上下文，涵盖框架版本、架构约定、组件规范、路由规则等关键约束

附录 C：依赖与版本明细

类别	依赖	版本
框架	next	16.2.9
框架	react / react-dom	19.2.4
国际化	next-intl	4.13.0
动画	framer-motion	12.40.0
数据校验	zod	4.4.3
图标	lucide-react	1.18.0
CSS 工具	clsx	2.1.1
CSS 工具	tailwind-merge	3.6.0
构建工具	tailwindcss	4.x
构建工具	@tailwindcss/postcss	4.x
类型系统	typescript	5.x
代码检查	eslint	9.x
代码检查	eslint-config-next	16.2.9
包管理	pnpm	11.7.0

附录 D：CI/CD 流水线配置

```
name: CI
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
```

```
jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: pnpm/action-setup@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 22
          cache: pnpm
      - run: pnpm install --frozen-lockfile
      - run: pnpm run lint
      - run: pnpm run typecheck
      - run: pnpm run build
```

流水线顺序严格遵循 lint → typecheck → build。push 到 main 自动触发 Production 部署，PR 创建或更新触发 Preview 部署（由 Vercel GitHub Integration 自动管理）。

附录 E：开发环境规格

项目	规格
处理器	11th Gen Intel Core i7-11700K @ 3.60 GHz
内存	16.0 GB RAM
GPU	NVIDIA GeForce RTX 3060 12 GB
操作系统	Windows 11 64-bit
Node.js	>= 22.0.0
pnpm	11.7.0
本地 AI 模型	Ollama（Qwen3 等开源模型）

参考资料

工具/框架	版本	用途
OpenCode Agent	—	AI 驱动开发 Agent
Ollama	—	本地 LLM 运行环境
Next.js	16.2.9	React 框架（App Router）
React	19.2.4	UI 库
TypeScript	5.x	类型系统
Tailwind CSS	4.x	CSS 框架
Framer Motion	12.40.0	动画库

工具/框架	版本	用途
next-intl	4.13.0	国际化框架
Zod	4.4.3	数据校验
pnpm	11.7.0	包管理器
Vercel	—	部署平台
GitHub Actions	—	CI/CD
Obsidian	—	文档管理
Git	—	版本控制

作者

KONG JUN SIENG

完成日期

2026-07-07