

Building a Portfolio with AI: A Documentation-First Approach

Executive Summary

0. 文档索引

1.1 个人品牌与网站展示文档

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 交互与动效说明

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

变更详情

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弹手册

VN游戏感改造执行手册

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: #FFB77C;

--color-primary-soft: #FFEEEC;

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.ai/docs access check

Context

73,760 tokens

37% used

\$0.00 spent

LSP

LSPs are disabled

▼ Todo

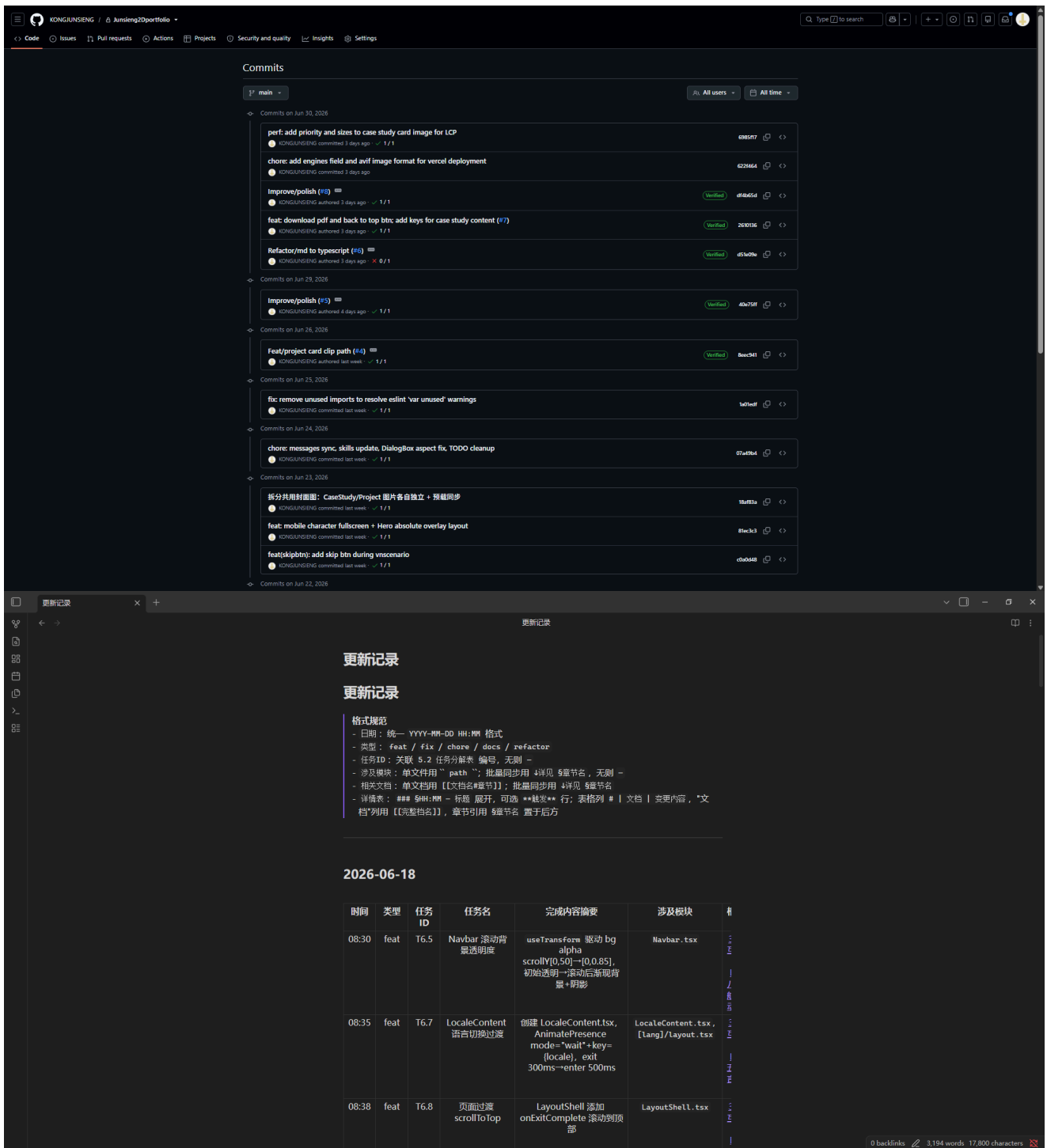
[✓] Fetch all relevant official docs pages (rules, themes, commands, skills, models, agents, github, providers, plugins)

[•] Write comprehensive updated file with all new sections

[] Output summary table of all changes

/~

OpenCode 1.15.13



This project is part of a job-seeking portfolio, aimed at HR professionals, technical managers, and fellow developers at tech companies. The final result is the JunsienPortfolio website (this site). What makes it stand out in both management and technology:

- It brings professional software engineering practices into a personal project, so every step can be traced back
- The AI Agent writes code based on documentation, while the human focuses on quality control, testing, translation, and bug fixes — creating a cycle where the human and Agent work together.

Background & Motivation

I chose my own portfolio site as the test case for two reasons:

- I wanted to build it
- It's the right size — big enough to cover a full project lifecycle (from requirements to deployment), but not too big

The main reasons I went with a **documentation-first** approach:

- To stop the AI Agent from writing code that doesn't match what I wanted, by setting clear rules and constraints through documentation
- To make the whole process repeatable, so I can use it as a template for future projects

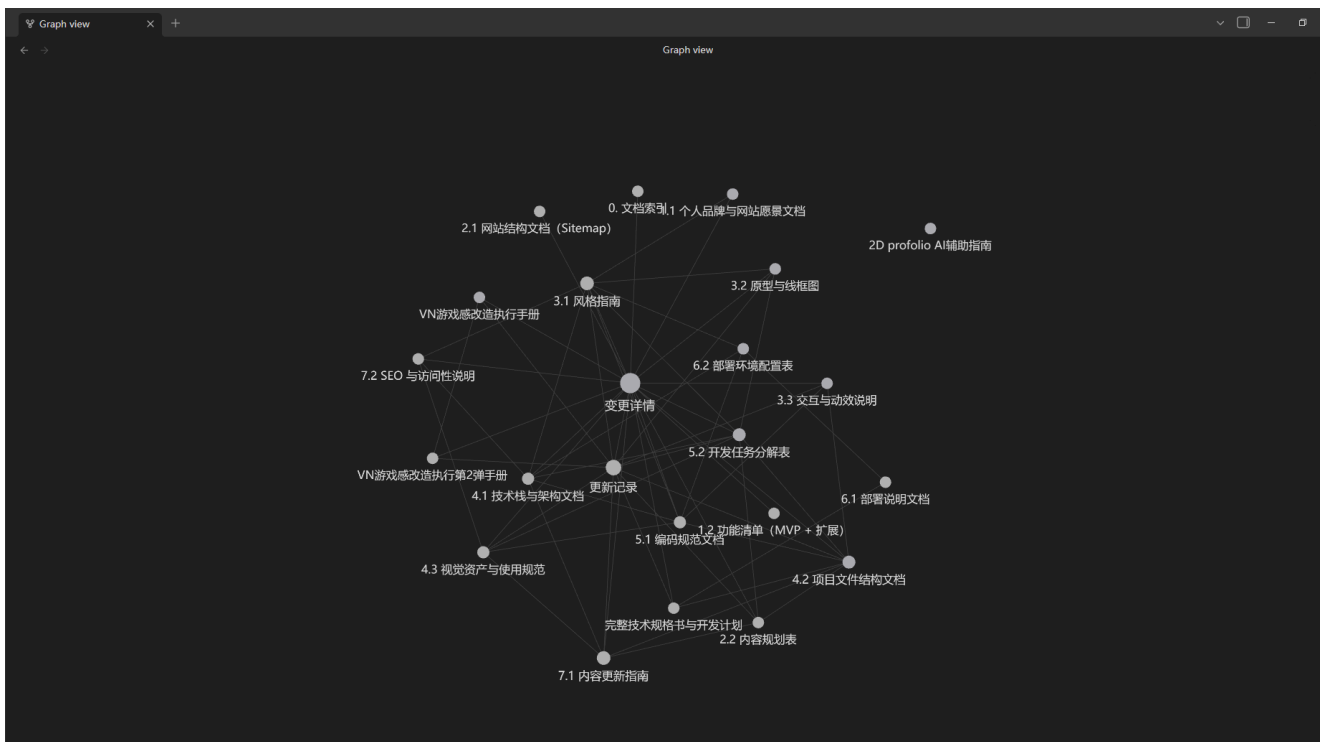
I chose the **OpenCode Agent** to try a new technical approach and see if documentation-driven development works in real projects. As for the tools:

- **Obsidian**: Uses Markdown, which is easy for an AI Agent to read but still clear for humans
- **Git**: Manages and versions both documents and project code in one place
- **Vercel**: My go-to free hosting solution

Methodology & Architecture

The Concept and Process of Documentation-First

Following software engineering standards, I set up a numbered document system **1.1–7.2**, covering the full lifecycle from brand vision to maintenance. The document relationship diagram shows how documents depend on each other.



Core documents include:

ID	Document Name	Description
1.1	Personal Brand & Site Vision	Positioning, goals, brand tone
1.2	Feature List (MVP + Extensions)	Feature scope and priorities
2.1	Site Structure (Sitemap)	Page structure and routing design
2.2	Content Planning Table	Page content and i18n key-value planning
3.1	Style Guide	Design tokens and UI specifications
4.1	Tech Stack & Architecture Document	Technology choices and architecture decisions
4.2	Project File Structure Document	Directory structure and component responsibilities
4.3	Visual Assets & Usage Guidelines	Image and font resource specifications
5.1	Coding Standards Document	Naming conventions, TS rules, conventions
5.2	Development Task Breakdown Table	105 tasks with breakdown and scheduling
6.1	Deployment Guide	Vercel Runbook and process
6.2	Environment Configuration Table	Node.js, pnpm version locking
7.1	Content Update Guide	Ongoing maintenance procedures
7.2	SEO & Accessibility Guide	SEO metadata and a11y specifications

(Full list in Appendix A). A dedicated VN Game-style Transformation Manual series covers the game-style design work. The whole process is managed with Obsidian + Git to keep version history.

Agent Configuration & Document Interaction

The `opencode.json` file sets up `external_directory` permissions, letting the Agent read the local Obsidian document library. `AGENTS.md` adds project context. In each development session, I manually point the Agent to the right documents so it understands the requirements, architecture, and task constraints. The Agent treats the documents as the single source of truth when writing code.

System Architecture

Layer	Choice	Notes
Framework	Next.js 16 App Router + React 19	Full TypeScript Strict
CSS	Tailwind CSS v4	Custom design tokens (@theme), no tailwind.config

Layer	Choice	Notes
Animation	Framer Motion	Sole animation library, game-style variants
i18n	next-intl	path-based ([lang]/...), trilingual (zh/ja/en)
Data Layer	Local JSON + Zod Validation	Zero CMS/Database, read-data.ts centralized reading
Fonts	Fully self-hosted	JP/EN: next/font/local, ZH: @font-face 13 subsets
UI	Custom components	No third-party UI library; game-style components
Package Manager	pnpm	CI uses --frozen-lockfile

The component architecture follows Server/Client Component separation: the root layout is a Server Component, while LayoutShell and LocaleContent are Client Components that handle animations and interaction states. (Full version details in Appendix C)

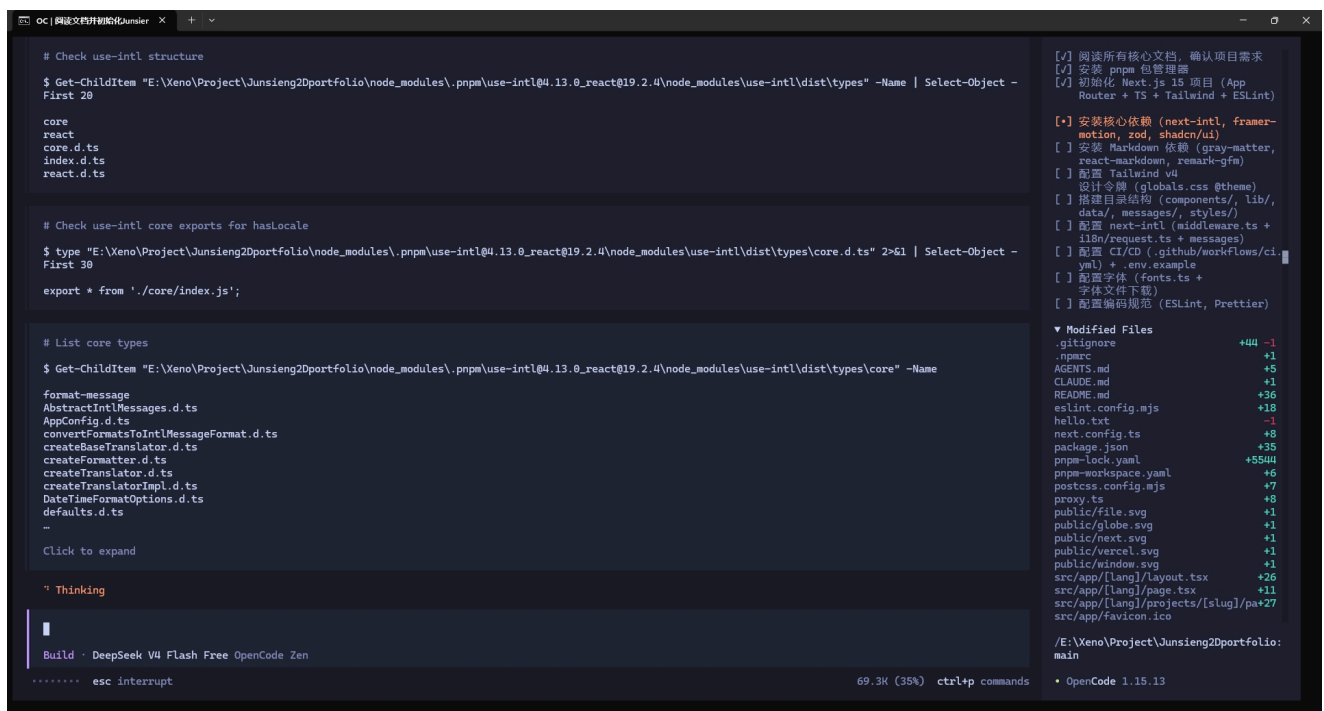
Deployment Plan

Dimension	Details
Platform	Vercel Hobby (free), domain jungsieng-portfolio.vercel.app
CI Tools	GitHub Actions, Node.js 22.x, pipeline: lint → typecheck → build
Auto-Trigger	main push → Production; PR create/update → Preview
Environment Variables	v1: none required
Rollback	Vercel Dashboard → Deployments → Promote to Production
Detailed Config	See Appendix D

Implementation Process

The development cycle took 13 days (June 16–30, 2026, about 2–3 hours per day), split into 7 phases using a documentation-driven, step-by-step approach. In each phase, development started after the OpenCode Agent read the relevant design documents. The Agent produced a first draft of the code, then I reviewed it, made changes, verified it, and synced the documents.

Phase 1: Project Scaffolding & Data Layer Setup (Day 1)



Set up the Next.js 16 App Router + TypeScript Strict + Tailwind v4 project structure, installed core dependencies, created the directory layout and custom design tokens. Made Zod schemas to validate data files, set up data files at the same time, configured international routing and a shared data access layer. Also set up the CI pipeline.

Phase 2: Global Layout & Core Homepage Sections (Day 2)

Built the global layout with a navigation bar, footer, and language switcher. Developed all five homepage sections in parallel — HeroSection (illustration + text dual column, stagger entry sequence), CaseStudiesSection, ProjectsSection, AboutSection, ContactSection. Set up dynamic routing and server-side SEO for Case Study detail pages. Fixed integration issues and passed all validation checks by the end of the phase.

Phase 3: Motion Enhancement & VN Game-style UI System (Day 3)

Made 4 groups of VN-style components and added them to each section. At the same time, completed motion upgrades like Hero parallax, navigation bar scroll gradient, and language switch transitions.

Phase 4: Intro Prologue System Development (Day 4–Day 6)

Built a full game-style prologue system — a four-stage state machine (opening animation → title screen → story playback → loading transition), with dialogue progression, choice branching, character portrait switching, mouse repulsion, auto-advance, and more. The trilingual scripts total 43 scenes with multiple branches and 6 endings.

Phase 5: Mobile Adaptation & Interaction Enhancement (Day 7–Day 10)

Redesigned mobile layout and interactions. Added expandable card details and idle animations. Also enriched project content and synced all three languages in parallel.

Phase 6: Code Cleanup & Document System Restructuring (Day 11–Day 12)

Cleaned out unused code and exports. Reorganized the change log into a two-layer structure with summary tables and detailed change tracking. Also updated the development task breakdown table.

Phase 7: Data Layer Restructuring & Deployment Preparation (Day 13)

Moved Case Study detail content from the Markdown compilation pipeline to reading directly from JSON message files, and rebuilt the template rendering system. Added a download button and a back-to-top button, configured image formats and engine limits. After passing all code checks, pushed to the main branch for Vercel deployment.

See the results table below for detailed metrics. Whenever the Agent's output drifted from what the documents specified, we used a two-way feedback loop of "fix code → sync documents" to keep things aligned. This created a repeated cycle of **human-AI collaboration**.

Key Challenges & Solutions

Challenge 1: Complex OpenCode Configuration

Description: Starting from scratch with the OpenCode Agent was a steep learning curve. Beyond the basics, advanced setup covered many areas: permission detail (`external_directory` to let the Agent read external Obsidian document libraries), skill reference injection, AGENTS.md project context, and token management. The official docs are thorough, but the documentation structure is extensive, so I had to learn the right order of settings, how parameters depend on each other, and which ones really matter through a lot of trial and error.

Resolution: Don't rely completely on AI chat tools or the official docs. AI chat tools (like ChatGPT, etc.) often give outdated or wrong guidance. Treat them like "better search engines" to find useful sources. The approach that worked: first look at other people's setup examples on GitHub to get a basic idea, then check chat tools and official docs to verify, and finally build your own full understanding through repeated practice.

Challenge 2: Time Investment in Requirements Discussion & Documentation

Description: The upfront cost is inherent to the documentation-first approach. Before writing any code, I spent about 8 hours and 25 minutes completing over 20 numbered documents, covering the full lifecycle from brand vision to operations. Each document went through detailed requirements discussion, solution evaluation, and item-by-item completion — not many revisions, but a big time investment.

Resolution: This comes down to a design choice that depends on the project type. The level of detail in the docs directly affects how much the AI Agent's output drifts from expectations — in this project, I had clear requirements and wanted high accuracy, so I chose to invest plenty of time in thorough docs. If the goal leans more toward fast prototype testing with AI leading creative exploration, you could reduce the doc effort and accept some drift, fixing it in later rounds.

Challenge 3: AI Agent Struggles to Proactively Identify Blind Spots

Description: AI Agents are built to follow user instructions. Most of the time, they just follow the user's line of thinking without spotting or pointing out gaps in the user's understanding. When users aren't experienced in a certain area, their logic can have incorrect assumptions or miss important points. For example, the Zod schema in this project was missing the `characterImage` field, which stopped character portraits from switching — I didn't realize the field needed to be declared, and the Agent didn't flag it either.

Resolution: There's no perfect fix for this given current AI model abilities and prompt precision. One useful way to mitigate this issue is to guide the Agent to switch roles through specific prompts — for example, asking it to "critique the current approach against industry standards" or "list common technical risks in this area that haven't been considered." How much this challenge affects the project depends on the Agent's intelligence, how clear the user's logic is, and how precise the prompts are.

Challenge 4: Humans Struggle to Precisely Describe UI Feel

Description: When tuning the UI, subjective things like spacing, color, and font choices are hard to turn into exact instructions an AI Agent can follow. Agents only understand specific numbers (px, rem, color codes), but people see interfaces as a whole — saying "this button doesn't stand out enough" or "the spacing between these two lines feels off" doesn't give the Agent something concrete to act on.

Resolution: Use a two-part approach based on whether a reference is available. If you have a reference, give concrete examples ("same spacing as the SectionTitle component" or "use

the button style from this website") to help the Agent build a frame of reference. If there's no reference and fine-tuning is needed, the developer should read and change the code directly — it's faster and gives better results than describing visual impressions over and over for the Agent to guess the intent.

Challenge 5: Deviation Between Documentation and Implementation

Description: What's written in design documents will inevitably drift from the actual code. The main reason is the natural change of requirements — during development, better solutions or new ideas come up, and the code moves away from the original docs. For example, the DialogBox component went through three design changes: CSS approach → PNG assets → CSS triple-layer rounded structure. Each change was done in code first, then the docs were updated afterward.

Resolution: Some drift is unavoidable; the key is having a reliable way to sync things back up. I chose a "fix code first, update docs later" workflow: first check that new solutions work in code, then update the design documents after confirming. This is a personal preference, not an industry rule — you could also choose a "fix documents first, then code" approach to keep docs in the lead. What matters is keeping them consistent in the end, not trying to perfectly prevent any drift.

Challenge 6: High Maintenance Cost of Document Synchronization

Description: After every code change, the related design documents need to be brought in line with the actual implementation. In this project, a single batch sync involved 6 to 22 documents, each needing a check on whether field descriptions, component interfaces, data structures, etc., matched the code. Later in the project, the 462-line change log was reorganized into a two-layer structure of summary tables and detailed change tracking to cut down the maintenance work.

Resolution: The maintenance work itself is driven by AI Agent prompts, so the time cost is manageable. The tricky part is that the human has to accurately record the context of each change so the Agent can fill in the details. I chose not to automate this sync process into AGENTS.md for three reasons:

- AGENTS.md instructions aren't stable enough
- It would take up context window space that might affect core tasks
- Automation could distract the Agent during important feature work

The level of detail in the documents isn't reduced — they still serve as key references for future maintenance and as a complete record of the work process.

Challenge 7: AI Agent Loses Focus in Long Sessions

Description: As sessions go on, the context window fills up, and the AI Agent's focus on the current task slowly drifts, leading to lower quality work. The Agent might forget earlier rules, confuse which files have been changed, or stray from the main requirements in complex tasks.

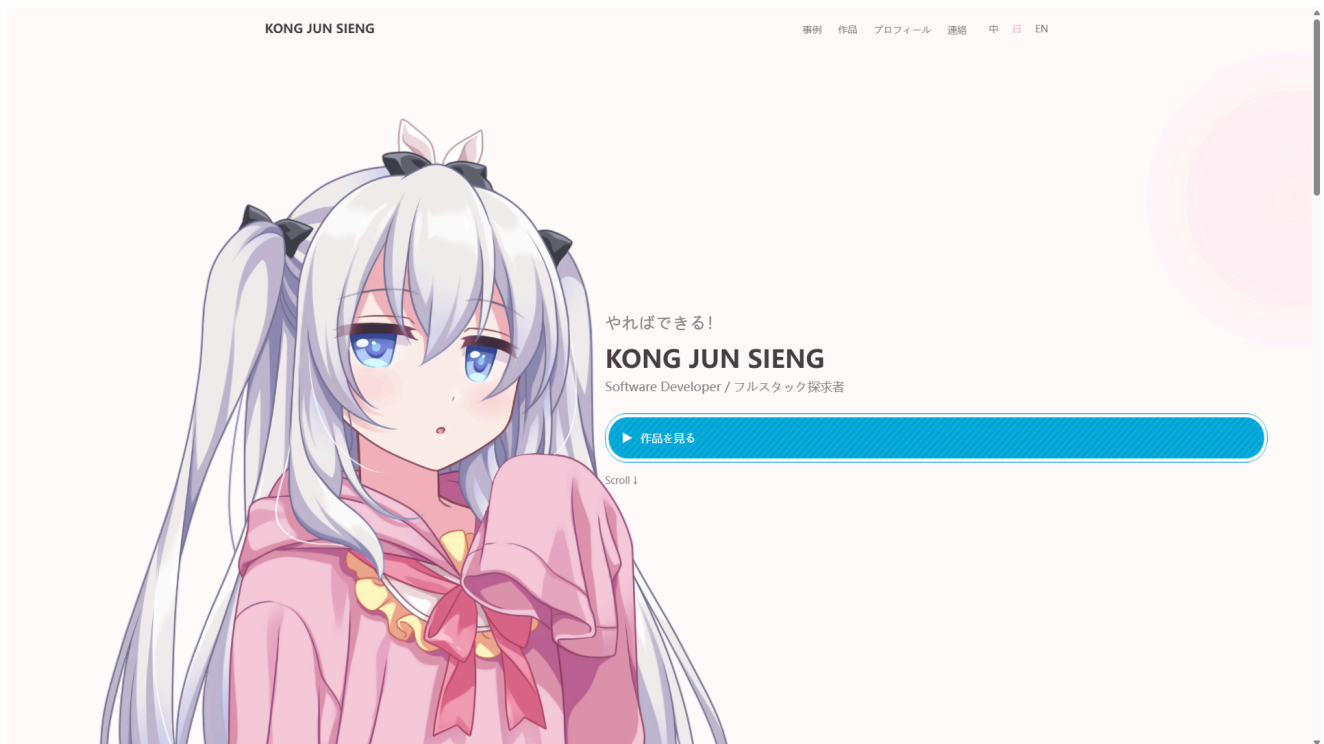
Resolution: Focus on one feature per session, close it when done, and start a new one. The OpenCode Agent runs with local Ollama models, so token usage isn't an issue — restarting sessions often doesn't cost anything extra. This approach costs more for paid Agent products that charge by token, but the community already has other fixes for those (like session summary compression, splitting work across sub-agents, etc.).

Results & Showcase

Deliverables Overview

Dimension	Result
Code Size	60+ files covering components, data layer, internationalization, styling, and configuration
Development Tasks	105 tasks (Phase 1–12), 100% completed
Design Documents	20+ numbered documents (1.1–7.2), including VN Game-style Transformation Manuals
Development Cycle	13 days (documentation phase about 8 hours 25 minutes)
Human-AI Role Distribution	Agent handles code generation and document sync; human handles defining requirements, reviewing quality, fixing bugs, and deploying

Features



Five Homepage Sections: HeroSection (full-screen character illustration + parallax + glow pulse animation), CaseStudiesSection (case study cards + idle wiggle effect), ProjectsSection (two-column grid + clip-path circular expand details), AboutSection (dialog box story + ATS skill tags), ContactSection (email/social/resume download, display only, no form).

KONG JUN SIENG

SOFTWARE DEVELOPER PORTFOLIO

▶ 設定

v1.0

▶ スキップ

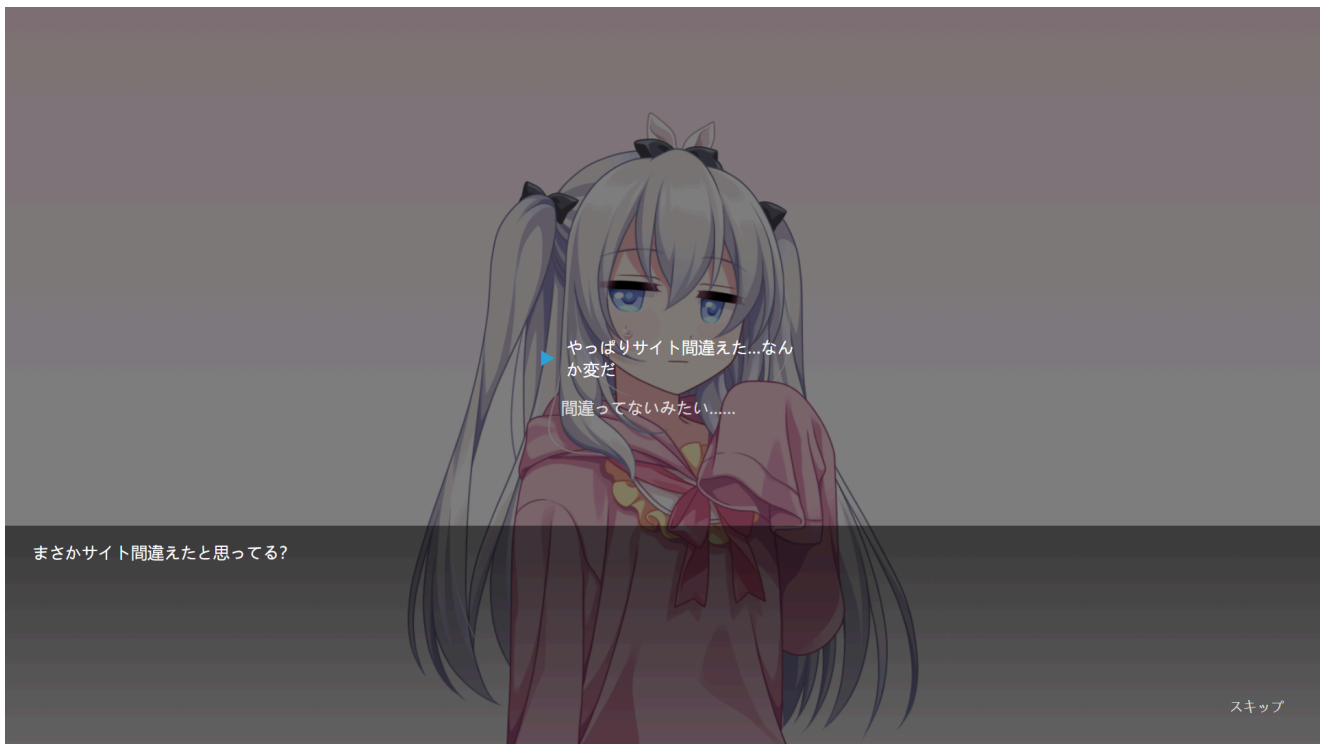


ルカちゃん

まさかサイト間違えたと思ってる？

▶ スキップ

VN Game-style Prologue System: Four-stage Intro flow (opening animation → title screen → story playback → loading transition), 43 scenes with trilingual scripts, 7 character expressions, multi-branch choices, mouse repulsion, auto-advance, and Skip. Visitors play through the prologue before reaching the homepage.



Game-style UI System: Chapter title system (CHAPTER label + blue underline), blue dialog box story, option button interactions, transition dialogues, dynamic navigation bar scroll effects, full Framer Motion variant suite (staggerContainer / chapterReveal / dialogSlideUp, etc.).

Trilingual Internationalization: zh/ja/en complete trilingual support, path-based routing (/[lang]/...), driven by next-intl.

Case Study Detail Page: 9-section structured content (executiveSummary → appendix), PDF download, BackToTop button, server-side generateMetadata SEO.

Code & Architecture Quality

- **Full TypeScript Strict:** tsconfig.json enables strict + noUncheckedIndexedAccess + noImplicitReturns, all component Props wrapped with Readonly<{...}>
- **Tailwind CSS v4 Design Tokens:** @theme defines a full color system (primary/secondary/accent/dialog-blue, etc.), fonts, spacing, border radius, shadows
- **Zod Data Validation:** All JSON data files pass schema type checking, no CMS or database
- **CI/CD:** GitHub Actions runs lint → typecheck → build, Node.js 22.x
- **Server/Client Component Separation:** Strict separation of server-side data reading and client-side animation interactions

Online Links

Production: <https://jungsieng-portfolio.vercel.app>

Discussion

Advantages of the Documentation-First + Agent Approach

The documentation-first approach paid off well — the first round of Agent generation already covered most of the feature structure. About a 40% increase in extra requirements (with more complexity) was the main thing that stretched the development timeline. This shows that the documentation-first model saves the most time in the early phase, and project delays come from natural requirement changes, not the process itself.

Another key benefit is that the Agent's output stayed stable when guided by documentation. Clear architecture rules, component interface definitions, and design token specs meant the Agent's code mostly met project standards on the first try. This cut down the back-and-forth that usually comes from re-explaining requirements and agreeing on conventions.

Natural Advantages of OpenCode CLI

OpenCode's command-line interface brings two unique advantages to the development workflow:

- **Multi-window Parallel Development:** OpenCode CLI is lightweight, so you can run multiple independent sessions at the same time, each working on different features (e.g., one window building components, another syncing documents) without them getting in each other's way. This cuts down serial waiting time, but each window's tasks need clear doc rules and quality standards — without a doc baseline, working in parallel across windows could lead to inconsistent code style or architecture.
- **Local Setup for Data Security:** OpenCode natively supports connecting to local Ollama models, which is great for projects with confidential info or private business logic. All the code and documents needed for Agent development can run fully locally without uploading anything to a third-party API service.

Inherent Limitations

This model struggles most with small UI tweaks. Things like spacing, color, and fonts are hard to turn into exact instructions the Agent can follow — Agents only understand numbers (px, rem, color codes), while people see the whole picture. The time spent describing and discussing often isn't worth the final result. When there's no reference to point to, the developer should just change the code directly — it's faster and works better than describing visual feelings over and over for the Agent to guess.

Workflow & Role Transformation

This model shifts the developer's role from "writing code" to "defining requirements, making decisions, and reviewing code." The developer no longer writes every line but instead turns requirements into clear documentation specs, makes technical choices at key points, and checks the quality and consistency of what the Agent produces.

This shift asks for stronger basic skills and industry experience. AI Agents are built to follow user instructions and won't spot or point out gaps in the user's understanding. When users lack experience in a certain area, their logic can have incorrect assumptions or miss important points that Agents can't automatically fix. A useful way to mitigate this is to guide the Agent to switch roles through prompt design (e.g., asking it to "critique the current approach against industry standards"), but the final quality still depends a lot on how clearly the user thinks and how deep their domain knowledge is.

Team Scenario Extrapolation

If this model were used in a team setting (each person paired with an Agent), things would likely work differently. The document discussion phase would need more communication because different roles (UI design, frontend, backend, testing) would need to coordinate. The Tech Lead would set up the overall framework first, then each role would refine their own area's documents. The development and testing phases would benefit from the fast start that documentation-driven work provides, with efficiency and quality depending on each role's document detail and accuracy. Note that this projection is based on personal project experience and hasn't been fully tested in a commercial team setting.

HR/Management Perspective

A candidate who actively tries AI Agent development approaches and builds a complete workflow is itself a positive sign. A possible question: "Why does it still take a lot of time even with Agent help?" — two reasons:

- Setting up and tuning the Agent has a learning curve; the results depend on how well the user knows the tool
- There's a direct trade-off between doc detail and output stability — the more detailed the docs (more upfront work), the more stable the Agent's output. Choosing between a fast start with lots of later fixes vs. a slower start with fewer fixes depends on the project's needs for delivery speed and quality

Open Source & Commercial Boundaries

The chosen tech stack (Next.js, React, Tailwind CSS, Framer Motion, Zod, etc.) all use permissive licenses (MIT, Apache 2.0, etc.) with no compliance issues for personal portfolio or commercial use. The OpenCode Agent is also open-source; companies using it in commercial work should review the license terms themselves.

Also, about the common worry that "developers who use AI Agents lack real skill": some people think that. But it's actually the opposite — the Agent's follow-instructions design means it relies heavily on the user's industry experience and logical judgment. The better someone is at using an Agent, the more it shows their skill in breaking down requirements, designing architecture, managing quality, and other core abilities.

Reflections & Best Practices

Documentation Density Over Iterative Fixing

The main takeaway confirms a principle: **spending enough time in the documentation phase to build a high-quality base gives much better long-term results than a "quick start → repeated fixes" cycle** (see "Key Challenges → Challenge 2" for specific data). This goes against the common advice of "fast prototype testing" — that approach is better for projects with unclear requirements or creative exploration, while this case shows that when developers have clear quality standards, putting time into documentation upfront is the most efficient path.

For a similar project next time, I'd keep the same level of documentation detail while trying more Agent configuration options (like skill chaining, multi-Agent division of work, etc.) to see how different setups affect output quality. Although I've tried to follow industry software engineering standards as much as possible, as someone without commercial team experience, the completeness and correctness of these standards need more testing in a real team setting — this also marks the boundaries of what I can do on my own and points out areas where I need to focus my learning.

Upgraded Understanding of Agent Development Capabilities

The process of setting up the OpenCode Agent from scratch gave me a full picture of the AI development toolchain. The way skill (adding domain expertise), MCP (Model Context Protocol), and AGENTS.md (project context) work together — skills give domain-specific knowledge, AGENTS.md adds project-level rules, MCP extends tool boundaries — forms the core structure of Agent configuration.

In prompt engineering, "role-shift prompting" proved to work well: asking the Agent to "critique the current approach against industry standards" or "list common technical risks in this area that haven't been considered" can partly fix the blind-spot problem that comes from the Agent's follow-instructions design. However, how well prompts work depends a lot on the user's own domain knowledge — users need to spot their own blind spots first before they can write prompts that guide the Agent to fill them in. In the age of AI-assisted development, the need for developers' basic skills isn't lowered — it's raised.

Practical Advice on Tool Selection

For anyone thinking about trying Agent-driven development, here's practical advice on choosing tools: if you have the budget and want the easiest start, go with Claude (Anthropic) — it's the most mature Agent product and a solid choice. If you want a low-cost option that can run locally, OpenCode is the best fit — it's open-source, so you can use local Ollama models for sensitive projects without uploading code or documents to third-party services. Starting Agent-driven development with OpenCode lets you learn core concepts like Agent configuration, document interaction, and session management in a zero-cost setup, making it easier to switch to paid products later.

Conclusion & Outlook

Core Findings

The documentation-first + AI Agent development model works well and pays off in personal projects. The core idea: use a professional software engineering documentation system (20+ numbered documents covering the full lifecycle) as the Agent's framework and single source of truth, creating a cycle where human strengths (defining requirements, making architecture decisions, reviewing quality) and Agent strengths (writing code, syncing documents, automating repetitive tasks) work together. In the first development session, this model produced about 80% of the feature structure at high quality in one go, confirming that more detailed documentation leads to better Agent output.

Looking at the bigger picture, the AI-assisted development era is reshaping what it means to be a good developer: while tool barriers are coming down, the need for breaking down requirements, making good architecture choices, and controlling quality is going up. The better someone is at using an Agent, the more it shows their strength in these core areas.

Applicable Scenarios

This model works best for:

- Developers who have fairly clear requirements and care more about delivery quality than fast prototype testing
- Projects of medium size (personal projects or small teams), where doc maintenance costs are manageable
- Developers with some software engineering background who can write structured design documents

For projects with unclear requirements, creative exploration goals, or very tight deadlines, going lighter on the docs for a faster start might be the more practical choice.

Future Directions

There are several areas to explore going forward. First, I plan to include Claude or other mainstream Agent products in comparison tests, checking how output quality and development speed differ across Agents when using the same documentation system. Second, I'll reuse this documentation-first + Agent-driven workflow in other personal projects (like a game development plan, still in the idea phase), to see if the approach transfers well to different areas. Finally, document workflow automation — if some documentation content can be auto-generated or synced through Agent or scripting tools, overall efficiency will go up even more. The shared goal: turning a single practice into a standard, reusable personal development approach.

Keywords

OpenCode Agent, documentation-first, Obsidian, Git, Vercel, Portfolio, Next.js, AI-driven development, human-AI collaboration, game-style UI, Ollama

Appendix

Appendix A: Complete Document System Inventory

ID	Document Name	Description
0	Document Index	Global document map and cross-references
1.1	Personal Brand & Site Vision	Positioning, goals, brand tone
1.2	Feature List (MVP + Extensions)	Feature scope and priorities
2.1	Site Structure (Sitemap)	Page structure and routing design
2.2	Content Planning Table	Page content and i18n key mapping
3.1	Style Guide	Design tokens and UI specifications
3.2	Prototypes & Wireframes	Layout and information hierarchy
3.3	Interaction & Motion Specifications	Animation specs and interaction behavior
4.1	Tech Stack & Architecture Document	Technology choices and architecture decisions
4.2	Project File Structure Document	Directory structure and component responsibilities
4.3	Visual Assets & Usage Guidelines	Image and font resource specifications
5.1	Coding Standards Document	Naming conventions, TS rules, conventions
5.2	Development Task Breakdown Table	105 tasks with breakdown and scheduling
6.1	Deployment Guide	Vercel Runbook and process
6.2	Environment Configuration Table	Node.js, pnpm version locking
7.1	Content Update Guide	Ongoing maintenance procedures
7.2	SEO & Accessibility Guide	SEO metadata and a11y specifications
—	VN Game-style Transformation Manual	VN-style transformation specifications and implementation record
—	VN Game-style Transformation Vol. 2	Intro prologue system specifications and implementation record
—	Complete Technical Specifications & Development Plan	Comprehensive technical specifications
—	AI-Driven Development Overview & Workspace Rules	Agent configuration and development workflow
—	Change Log	Change log summary (108 lines)
—	Change Details	Detailed change tracking (394 lines)
—	TODO Development Checklist	Phased checklist

Appendix B: OpenCode Agent Configuration Highlights

opencode.json core configuration:

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

- **external_directory** : Gives the Agent permission to read the Obsidian document library, so it can directly access all 20+ design documents during development sessions
- **edit.ask** : Changes to Obsidian documents need manual approval, stopping the Agent from changing design documents without review
- **instructions** : Loads the frontend-design skill, adding frontend design decision guidance for the Agent
- **AGENTS.md**: Adds project context from the project root, covering framework version, architecture conventions, component specs, routing rules, etc.

Appendix C: Dependency & Version Details

Category	Dependency	Version
Framework	next	16.2.9
Framework	react / react-dom	19.2.4
Internationalization	next-intl	4.13.0
Animation	framer-motion	12.40.0
Data Validation	zod	4.4.3
Icons	lucide-react	1.18.0
CSS Utilities	clsx	2.1.1
CSS Utilities	tailwind-merge	3.6.0
Build Tool	tailwindcss	4.x
Build Tool	@tailwindcss/postcss	4.x
Type System	typescript	5.x
Linting	eslint	9.x
Linting	eslint-config-next	16.2.9

Category	Dependency	Version
Package Manager	pnpm	11.7.0

Appendix D: CI/CD Pipeline Configuration

```
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
```

The pipeline follows the order lint → typecheck → build. Pushes to main automatically trigger a Production deployment, while PR creation or updates trigger a Preview deployment (managed by Vercel GitHub Integration).

Appendix E: Development Environment Specifications

Item	Specification
Processor	11th Gen Intel Core i7-11700K @ 3.60 GHz
Memory	16.0 GB RAM
GPU	NVIDIA GeForce RTX 3060 12 GB
OS	Windows 11 64-bit
Node.js	>= 22.0.0
pnpm	11.7.0
Local AI Model	Ollama (Qwen3 and other open-source models)

References

Tool/Framework	Version	Purpose
OpenCode Agent	—	AI-driven development Agent
Ollama	—	Local LLM runtime environment
Next.js	16.2.9	React framework (App Router)
React	19.2.4	UI library
TypeScript	5.x	Type system
Tailwind CSS	4.x	CSS framework
Framer Motion	12.40.0	Animation library
next-intl	4.13.0	Internationalization framework
Zod	4.4.3	Data validation
pnpm	11.7.0	Package manager
Vercel	—	Deployment platform
GitHub Actions	—	CI/CD
Obsidian	—	Document management
Git	—	Version control

Author

KONG JUN SIENG

Completed

2026-07-07