

ドキュメントをもとにした人間とAIの協業：AI Agent開発の方法論

概要

0. 文档索引

1.1 个人品牌与网站搭建文档

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

层画

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;

25 backlinks 4,435 words 21,597 character

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: 400ms

% 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

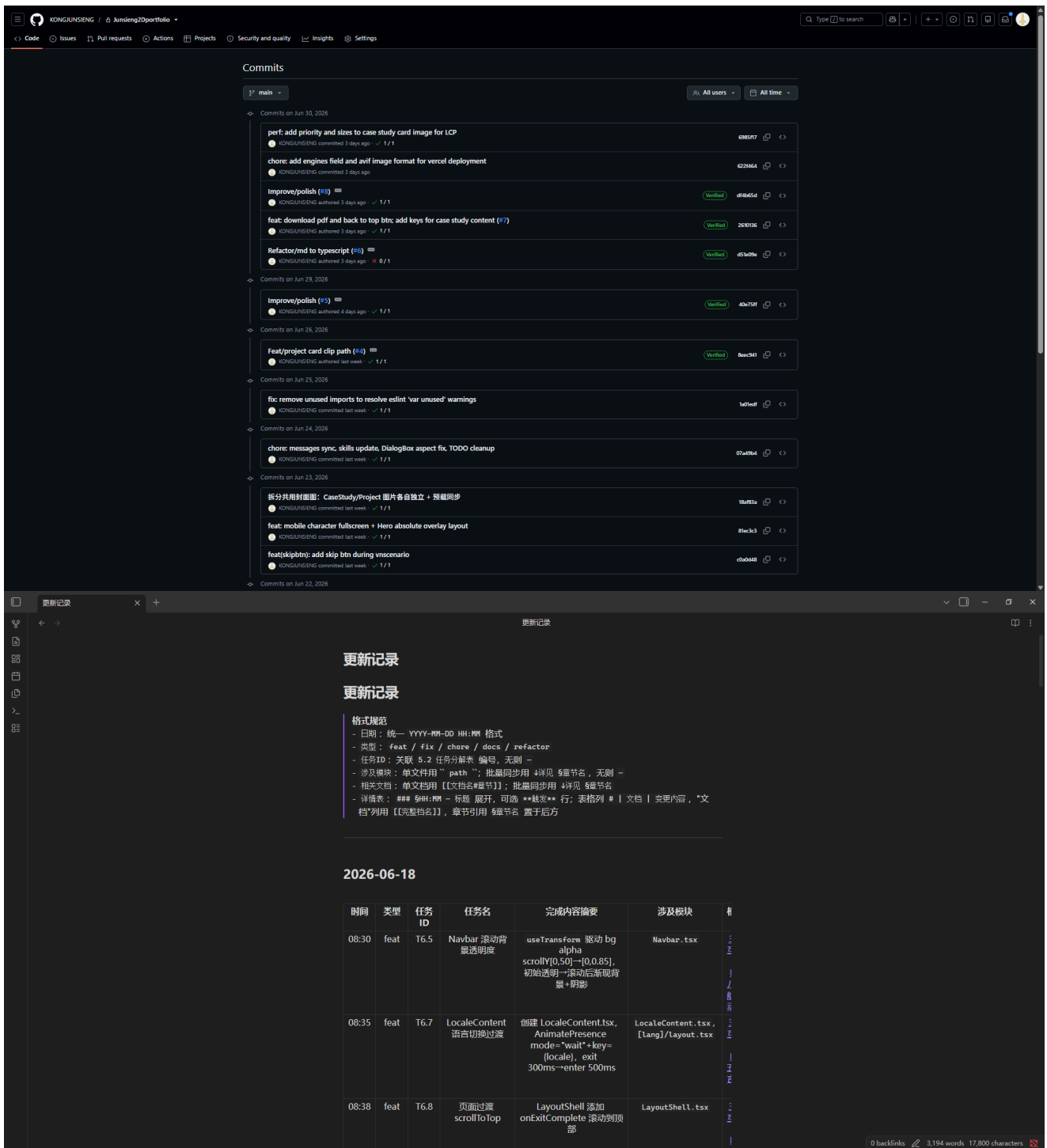
[✓] Fetch all relevant official docs pages (rules, themes, commands, skills, models, agents, github, providers, plugins)

[•] Write comprehensive updated file with all new sections

[] Output summary table of all changes

/~

OpenCode 1.15.13



このプロジェクトは求職用ポートフォリオの一部で、テクノロジー企業のHR、テクニカルマネージャー、開発者を対象としています。最終成果物はJunsienGPortfolioのショーケースWebサイト（このサイト）です。このプロジェクトの特徴は、次の2点です：

- ソフトウェア工学の標準的なドキュメントの作り方と進め方を個人プロジェクトに取り入れ、すべての工程をあとから追跡できるようにした
- AI Agentがドキュメントをもとにコードを生成し、人間は品質管理、テスト、翻訳、バグ修正に集中する協力のサイクルを作った

背景と動機

個人ポートフォリオサイトを実験の場にした理由は二つあります：

- 作りたかったから

- プロジェクトの規模がちょうどよく、要件からデプロイまでの流れを一通り体験できるから

ドキュメントファーストのやり方をとった中心的な理由：

- AI Agentが意図から外れたコードを生成するのを防ぎ、ドキュメントではっきりした制約と方向性を示すため
- プロセス全体を再現可能にし、将来のプロジェクトで参考にできるようにするため

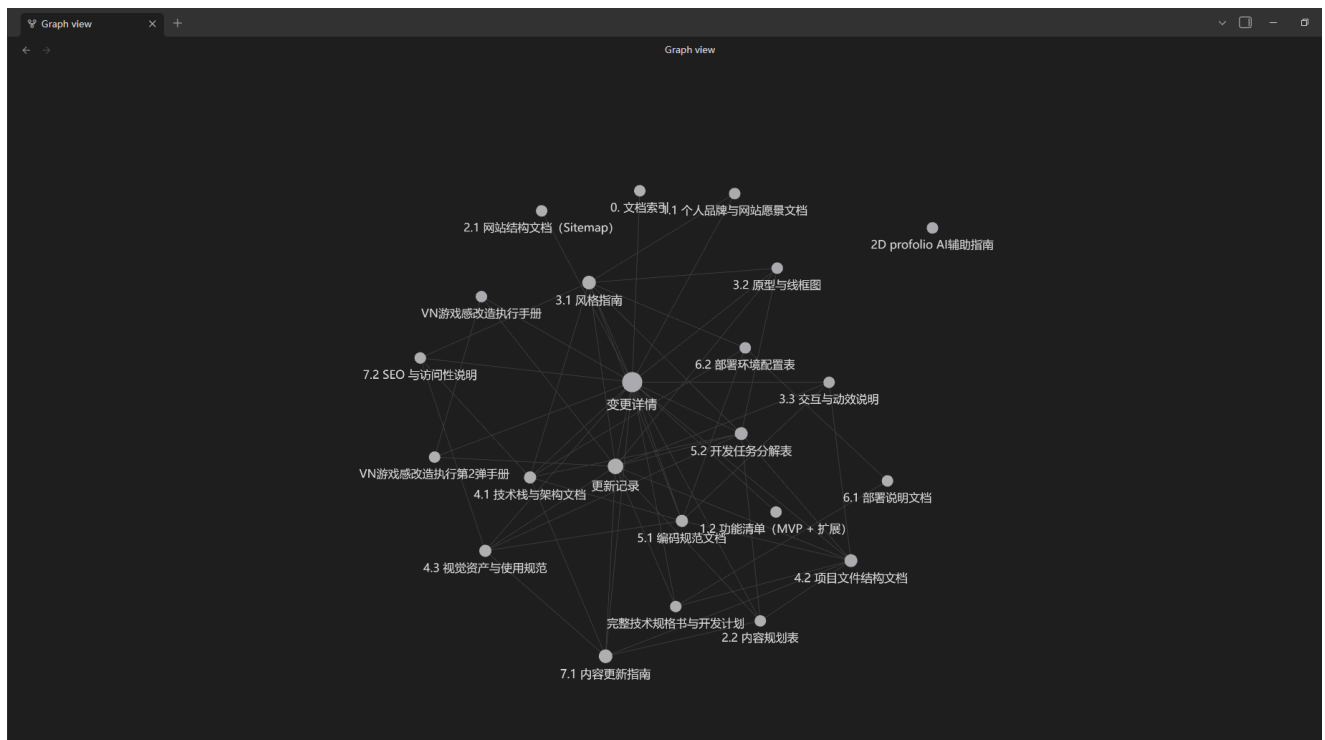
OpenCode Agentを選んだのは、新しい技術の進め方を探りながら、実際のプロジェクトでドキュメントファーストのやり方がどれだけ使えるかを確かめたかったからです。ツールの選び方については：

- Obsidian**：Markdown形式で書けるので、AI Agentが解析しやすく、人間も読みやすい
- Git**：ドキュメントとプロジェクトコードをまとめて管理でき、バージョン管理もできる
- Vercel**：無料でデプロイできるのが一番の魅力

方法論とアーキテクチャ

ドキュメントファーストの考え方と進め方

ソフトウェア工学の標準的なやり方に沿って、番号をつけたドキュメントの体系 **1.1–7.2** を作り上げました。ブランドのビジョンから運用保守まで、開発の全体の流れをカバーします。ドキュメント同士の関係図で、依存関係がわかるようになっています。



主なドキュメントは次の通りです：

ID	ドキュメント名	説明
1.1	個人ブランドとサイトビジョン	ポジショニング、目標、ブランドトーン

ID	ドキュメント名	説明
1.2	機能一覧（MVP + 拡張）	機能範囲と優先順位
2.1	サイト構造（サイトマップ）	ページ構造とルーティング設計
2.2	コンテンツ計画表	ページコンテンツとi18nキー計画
3.1	スタイルガイド	デザイントークンとUI仕様
4.1	技術スタックとアーキテクチャ	技術の選び方とアーキテクチャの決定
4.2	プロジェクトファイル構造	ディレクトリ構造とコンポーネントの役割
4.3	ビジュアルアセットと使用ガイドライン	画像とフォントリソースの仕様
5.1	コーディング規約	命名規則、TSルール、規約
5.2	開発タスク分解表	105タスクの分解とスケジューリング
6.1	デプロイガイド	Vercelでの手順とプロセス
6.2	環境設定表	Node.js、pnpmバージョン固定
7.1	コンテンツ更新ガイド	継続的な運用手順
7.2	SEOとアクセシビリティ	SEOメタデータとa11y仕様

（完全な一覧は付録Aを参照）。VNゲーム風改造マニュアルシリーズは専門ブランチとして別に作成しました。すべてのプロセスはObsidian + Gitで管理し、バージョンをあとから追跡できるようにしています。

Agentの設定とドキュメントの連携

`opencode.json` で `external_directory` の権限を設定し、Agentが手元のObsidianドキュメントを読み取れるようにします。`AGENTS.md` でプロジェクトの情報を渡します。開発セッションごとに関連ドキュメントを手動で参照させ、Agentに要件、アーキテクチャ、タスクの制約を理解させます。Agentはコードを生成するとき、ドキュメントを唯一の情報源として使います。

システムアーキテクチャ

レイヤー	選択	備考
フレームワーク	Next.js 16 App Router + React 19	完全TypeScript Strict
CSS	Tailwind CSS v4	カスタムデザイントークン（@theme）、tailwind.configなし
アニメーション	Framer Motion	唯一のアニメーションライブラリ、ゲーム風バリエーション
i18n	next-intl	パスベース（/[lang]/...）、三ヶ国語（zh/ja/en）

レイヤー	選択	備考
データ層	ローカルJSON + Zod検証	CMSやデータベースは不要、read-data.tsでまとめて読み取り
フォント	すべてセルフホスト	JP/EN: next/font/local、ZH: @font-face 13サブセット
UI	カスタムコンポーネント	サードパーティのUIライブラリは使わず、ゲーム風コンポーネント
パッケージ管理	pnpm	CIは--frozen-lockfileを使用

コンポーネントの構成はServer ComponentとClient Componentの境界に従います：ルートレイアウトはServer Component、LayoutShellとLocaleContentはアニメーションやインタラクションの状態を扱うClient Componentです。（詳細は付録Cを参照）

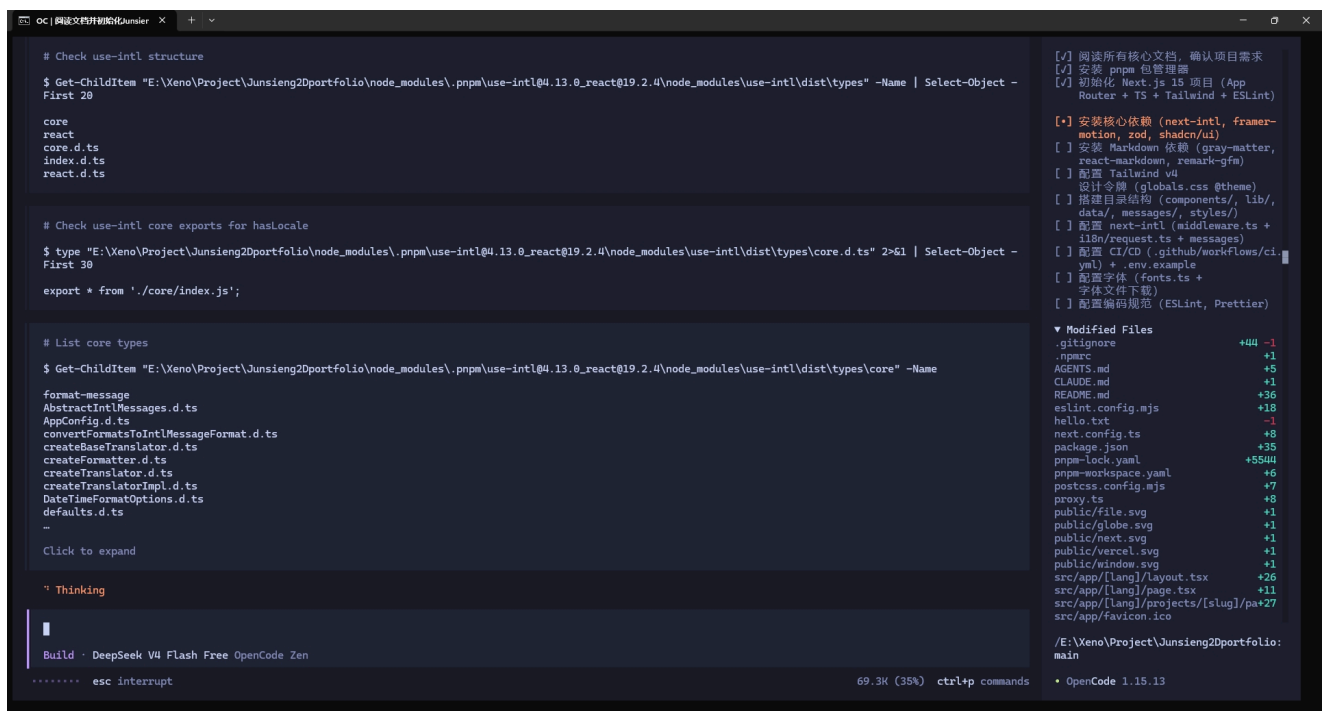
デプロイ計画

項目	詳細
プラットフォーム	Vercel Hobby（無料）、ドメイン jungsieng-portfolio.vercel.app
CIツール	GitHub Actions、Node.js 22.x、パイプラインの順番：lint → typecheck → build
自動トリガー	mainにプッシュ → 本番環境、PR作成/更新 → プレビュー環境
環境変数	v1：不要
ロールバック	Vercel Dashboard → Deployments → Promote to Production
詳細設定	付録Dを参照

実装の流れ

開発期間は13日間（2026年6月16日～30日、毎日約2～3時間）で、ドキュメントをもとに段階的に進め、7つのフェーズに分けました。各フェーズでは、OpenCode Agentが対応する設計ドキュメントを読んでから開発を始めます。Agentがコードの最初の案を生成し、人間がレビュー、修正、確認、ドキュメントの同期を行います。

フェーズ1：プロジェクトの土台とデータ層の構築（1日目）



Next.js 16 App Router + TypeScript Strict + Tailwind v4のプロジェクトの土台を初期化し、コアの依存関係をインストール、ディレクトリ構造とカスタムデザイントークンを設定しました。Zodスキーマを作ってデータファイルに制約をかけ、データファイルを同時に作りながら、国際化ルーティングとデータを読み取る層を設定。CIパイプラインも構築しました。

フェーズ2：全体レイアウトとホームページの主要セクション（2日目）

ナビゲーションバー、フッター、言語切替機能を備えた全体レイアウトを作成。5つのホームページセクションを同時に開発しました — HeroSection（イラスト+テキストの二段組み、順番に表示）、CaseStudiesSection、ProjectsSection、AboutSection、ContactSection。ケーススタディ詳細ページには動的ルーティングとサーバーサイドSEOを入れました。統合の問題を修正し、フェーズ終了時にすべての確認を通過しました。

フェーズ3：動きの強化とVNゲーム風UIシステム（3日目）

4つのVN風コンポーネントグループを作り、各セクションに組み込みました。Heroのパララックス、ナビゲーションバーのスクロールによるグラデーション変化、言語切替のアニメーションなどの動きの強化も同時に完了しました。

フェーズ4：はじめのプロローグシステムの開発（4日目～6日目）

完全なゲーム風プロローグシステムを作りました。4段階の状態管理（オープニングアニメーション → タイトル画面 → シナリオ再生 → ローディング画面への切り替え）に加えて、ダイアログの進行、選択肢による分岐、キャラクターポートレートへの切り替え、マウスを避ける動き、自動進行などの操作に対応しています。三ヶ国語のスクリプトは全部で43シーンあり、複数の分岐と6つのエンディングがあります。

フェーズ5：モバイル対応と操作感の強化（7日目～10日目）

モバイル向けのレイアウトと操作感を見直しました。カードの展開表示と、何もしていないときのアニメーションを追加。プロジェクトの内容を充実させると同時に、三ヶ国語の同期も行いました。

フェーズ6: コードの整理とドキュメント体系の再構築（11日目～12日目）

使っていないコードとエクスポートを体系的に削除しました。変更ログを、要約の表と詳しい変更の記録の二層構造に整理し直しました。同時に開発タスク分解表も改訂しました。

フェーズ7: データ層の再構築とデプロイの準備（13日目）

ケーススタディの詳細内容を、MarkdownからメッセージJSONを直接読み取る方式に切り替え、表示の仕組みを再構築しました。ダウンロードボタンとトップへ戻るボタンを追加し、画像の形式とエンジンの制約を設定。コードを完全に確認したあと、mainブランチにプッシュしてVercelへのデプロイ準備を整えました。

詳しい数値については下の成果表を参照してください。Agentの動きがドキュメントの想定からずれた場合は、そのつど「コードを直す → ドキュメントを同期する」という双方向のフィードバックで修正し、**人間とAIが協力して繰り返し改善するサイクル**を作りました。

主要な課題と解決策

課題1: OpenCodeの複雑な設定

説明: OpenCode Agentをゼロから使い始めるとき、設定の学習にかなりの時間がかかりました。基本機能に加えて、高度な設定にはいくつもの要素があります：権限の細かさ（`external_directory` でAgentが外部のObsidianドキュメントを読み取れるようにする）、スキルの参照、AGENTS.mdでのプロジェクト情報の定義、トークン管理など。公式ドキュメントは詳しいですが構造が大きく、設定の順番やパラメータ同士の影響、各設定が本当に必要かどうかの判断に、たくさんの試行錯誤が必要でした。

対策: AIチャットツールや公式ドキュメントだけに頼るのは避けました。AIチャットツール（ChatGPTなど）が教える設定方法は、古かったり間違っていたりすることがよくあります。これらは「高性能な検索エンジン」として使い、情報の出どころを見つけるために使いました。実際の学習の流れ：まずGitHubなどで他の人の設定例を参考にして大まかに理解し、次にチャットツールと公式ドキュメントを組み合わせ確認しながら操作し、最後に繰り返し試して自分の設定への理解を深めました。

課題2: 要件の検討とドキュメント作成にかかる時間

説明: ドキュメントファーストのやり方には、はじめにどうしても時間がかかります。コードを書き始める前に、約8時間25分かけて20以上の番号付きドキュメントを一つずつ作りま

した。ブランドのビジョンから運用まで、開発の全体の流れをカバーしています。各ドキュメントは要件を詳しく検討し、解決策を評価し、項目ごとに書き込んでいきました。修正の回数は少ないですが、時間は大きくかかっています。

対策： これはプロジェクトの性質によって判断すべき設計上の選択です。ドキュメントの細かさは、AI Agentの出力と期待とのズレの大きさに直接影響します — このプロジェクトでは開発者が明確な要件を持ち、高い正確さを求めたため、十分な時間をかけて詳しいドキュメントを作ることになりました。プロジェクトの目標が素早い試作検証や、AI主導で自由に発想を広げることにある場合は、ドキュメントへの投資を減らし、あとで修正しながら進める方法も有効です。

課題3：AI Agentが知識の不足を自分から指摘できない

説明： AI Agentは基本的に、ユーザーの指示に従うように作られています。たいていの場合はユーザーの考えに沿うだけで、ユーザーの理解が足りない部分を自分から見つけて指摘することはしません。ユーザーが特定の分野に経験がない場合、その考え方自体に標準的でない思い込みや、考えるべきことの抜けがあるかもしれません。例えば、このプロジェクトでZodスキーマに `characterImage` フィールドがなかったため、キャラクターポートレートの切り替えができませんでした — ユーザーはこのフィールドが必要だと気づかず、Agentも自分から警告しませんでした。

対策： 今のAIモデルの性能やプロンプトの正確さの範囲では、完全な解決方法はありません。効果的な対処法は、特定のプロンプトの工夫でAgentの役割を切り替えさせることです — 例えば「業界の標準と比べて、今のやり方の問題点を指摘してください」や「この分野でよくあるけど、まだ考えられていない技術的なリスクを挙げてください」と頼むことです。Agentの知能のレベル、ユーザーの考えの明確さ、プロンプトの正確さの3つが、この課題の影響の大きさを決めます。

課題4：人間がUIの感覚を正確に説明できない

説明： UIの細かい調整では、間隔、色合い、文字の見た目などの主観的な感覚を、AI Agentが実行できる正確な指示に数値化するのが難しいです。Agentは具体的な数値（px、rem、色コード）しか理解できませんが、人間のインターフェースに対する印象は全体としてのものです — 「このボタンが十分に目立たない」「この2行のテキストの間隔がおかしい」といった説明は、Agentには実行しづらいものです。

対策： 参考にするものがあるかどうかで、2つの方法に分けました。参考がある場合は具体的な例を示します（「SectionTitleコンポーネントと同じ間隔」「特定のWebサイトのボタンのスタイルを参考に」）。これでAgentに基準を与えます。参考になるものがなくて細かい調整が必要な場合は、開発者が直接コードを読んで直す方が、自然言語で見た目の印象を何度も説明してAgentに意図を推測させるよりも、効率も結果もよいです。

課題5: ドキュメントと実際のコードとのズレ

説明: 設計ドキュメントの仕様と、実際のコードの間には、どうしてもズレが生まれます。根本的な原因は主に、開発中に要件が自然に変わっていくことです — よりよい解決策や新しいアイデアが浮かび、コードが元のドキュメントの仕様からずれていきます。例えば、DialogBoxコンポーネントはCSS案 → PNG画像 → CSSの三層構造の丸い角、という3回の設計の変更を経ており、毎回コードを先に作ってから、あとでドキュメントを同期しました。

対策: ズレは避けられません。大切なのは、ちゃんと同期できる仕組みを作ることです。このプロジェクトでは「コードを先に直し、あとでドキュメントを同期する」という進め方を選びました: 新しい解決策が本当に使えるかをコードで先に確かめ、確認できたら変更に対応する設計ドキュメントに反映します。これは個人の好みであり、業界の標準というわけではありません — 「ドキュメントを先に直し、あとでコードを直す」という流れを選んで、ドキュメントを常に先行させることもできます。大事なのは、最終的に両者が合っている状態を保つことであって、ズレを絶対に起こさないようにすることではありません。

課題6: ドキュメントの同期にかかる手間

説明: コードを変更するたびに、対応する設計ドキュメントを実際のコードと合わせる必要があります。このプロジェクトでは、1回のまとまった同期で6〜22のドキュメントを扱い、各ドキュメントのフィールドの説明、コンポーネントのインターフェース、データ構造などがコードと合っているかを確認する必要がありました。プロジェクトの後半では、462行の更新記録を、要約の表と詳しい変更の記録の二層構造に整理して、メンテナンスの負担を減らしました。

対策: メンテナンス作業自体はAI Agentへの指示で行い、時間は管理できる範囲でした。大事な課題は、人間がAgentに正しい変更内容を伝えるために、正確な変更の記録を残す必要があることです。ドキュメント同期の要件をAGENTS.mdに自動化しなかった理由は3つあります:

- AGENTS.mdの指示が安定して動くとは言い切れなかった
- コンテキストウィンドウを占有して、本来のタスクに影響する可能性があった
- 自動化することでAgentが大事な機能開発で集中力を失う可能性があった

ドキュメントの細かさは減らしません — ドキュメントは将来のメンテナンスで重要な参考資料であり、作業のプロセスを完全に残した証拠でもあります。

課題7: 長いセッションでAI Agentの集中が続かない

説明： セッションが長くなるにつれて、それまでのやりとりがたまっていき、AI Agentの今のタスクへの集中がだんだん薄れて、作業の質が落ちます。Agentは最初に伝えた制約を忘れたり、変更したファイルの状態を混同したり、複雑なタスクで本来の要件からずれたりすることがあります。

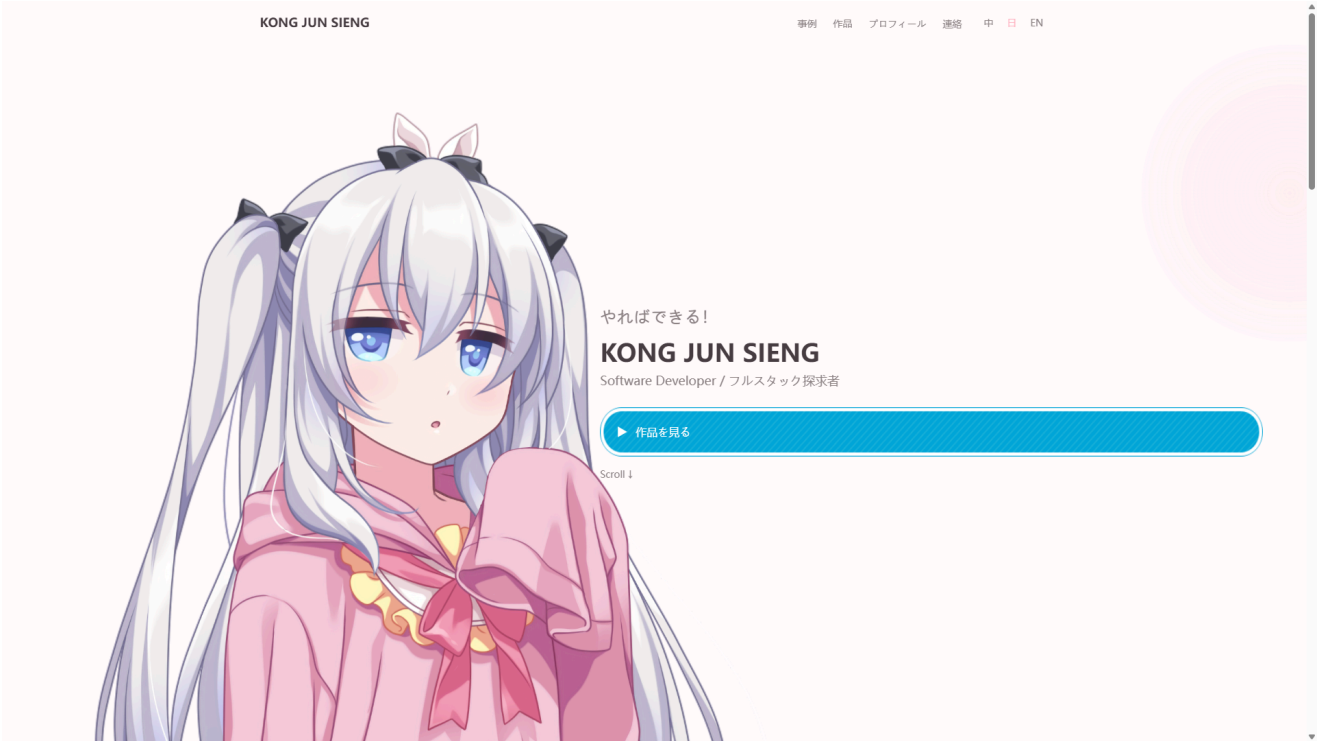
対策： 1回のセッションでは1つの機能に集中し、終わったら閉じて新しいセッションを始めます。OpenCode AgentはローカルのOllamaモデルで動くため、トークンの消費は気になりません — セッションを頻繁に再開しても追加コストはかかりません。この方法はトークン課金の有料Agent製品には経済的ではないですが、コミュニティにはすでに他の解決策（セッションの要約圧縮、サブエージェントの分担など）があります。

成果と展示

成果物概要

項目	結果
コードの規模	60以上のファイル（コンポーネント、データ層、国際化、スタイリング、設定をカバー）
開発タスク	105タスク（フェーズ1～12）、100%完了
設計ドキュメント	20以上の番号付きドキュメント（1.1～7.2）、VNゲーム風改造マニュアルを含む
開発期間	13日間（ドキュメント作成は約8時間25分）
人間とAIの役割分担	Agentがコード生成とドキュメント同期を担当、人間が要件定義、品質レビュー、バグ修正、デプロイを担当

機能



5つのホームページセクション: HeroSection (画面いっぱいのキャラクターイラスト + パララックス + 光るパルスアニメーション)、CaseStudiesSection (ケーススタディカード + 揺れアニメーション)、ProjectsSection (二段組みグリッド + clip-pathで円形に広がる詳細表示)、AboutSection (ダイアログボックス形式の自己紹介 + ATSスキルタグ)、ContactSection (メール/ソーシャルリンク/履歴書ダウンロード、表示のみでフォームなし)。



VNゲーム風プロローグシステム: 4段階の導入フロー (オープニングアニメーション → タイトル画面 → シナリオ再生 → ローディング画面への切り替え)、三ヶ国語の43シーン、7キャラクターの表情、複数の分岐選択肢、マウスを避ける動き、自動進行、スキップ機能。訪問者はホームページに入る前にプロローグを体験します。



ゲーム風UIシステム：チャプタータイトルシステム（CHAPTERラベル + 青い下線）、青いダイアログボックスでの説明、選択肢ボタンの操作、切り替わる台詞、動くナビゲーションバーのスクロール効果、Framer Motionのバリエーション揃い（staggerContainer / chapterReveal / dialogSlideUpなど）。

三ヶ国語対応：zh/ja/enの完全三ヶ国語サポート、パスベースのルーティング（/[lang]/...）、next-intlで動作。

ケーススタディ 詳細ページ：9セクションの構造化された内容（executiveSummary → appendix）、PDFダウンロード、BackToTopボタン、サーバーサイドのgenerateMetadata SEO。

コードとアーキテクチャの品質

- **完全なTypeScript Strict：**tsconfig.jsonでstrict + noUncheckedIndexedAccess + noImplicitReturnsを有効にし、すべてのコンポーネントのPropsはReadOnly<{...}>でラップ
- **Tailwind CSS v4のデザイントークン：**@themeで色のシステム（primary/secondary/accent/dialog-blueなど）、フォント、余白、角丸、影を定義
- **Zodによるデータ検証：**すべてのJSONデータファイルがスキーマの型チェックを通り、CMSやデータベースは不要
- **CI/CD：**GitHub Actionsでlint → typecheck → build、Node.js 22.x
- **Server/Client Componentの区別：**サーバー側のデータ読み取りとクライアント側のアニメーションや操作をきちんと分離

オンラインリンク

本番環境： <https://jungsieng-portfolio.vercel.app>

議論

ドキュメントファースト + Agentのやり方の利点

ドキュメントファーストの初期投資の効果はとても大きいです — 最初のAgent生成で、機能の大枠のほとんどをカバーできました。もとのドキュメントから約40%の要件が増えて複雑さが増したことが、開発期間が延びた主な原因です。これは、ドキュメントファーストのモデルの時間的な有利さは初期の段階で特に大きく、プロジェクトの遅れの原因はやり方そのものではなく、要件が自然に変わっていくことにあることを示しています。

ドキュメントで制約をかけたときのAgentの出力の安定性も、もう一つの中心的な利点です。はっきりしたアーキテクチャのルール、コンポーネントのインターフェースの定義、デザイントークンの仕様があることで、Agentが生成したコードが最初の時点でほぼプロジェクトの基準を満たし、要件を何度も確認したりルールを調整するためのやりとりの手間を減らしました。

OpenCode CLIの強み

OpenCodeのコマンドラインインターフェースの設計には、開発の進め方において2つのユニークな利点があります：

- **複数ウィンドウでの同時開発：** OpenCode CLIは軽い設計なので、複数の独立したセッションを同時に実行できます。それぞれのセッションは違う機能に集中でき（例：一方のウィンドウでコンポーネント開発、もう一方でドキュメント同期）、お互いに干渉しません。これで待ち時間を減らしますが、各ウィンドウのタスクにはっきりしたドキュメントの制約と品質基準が必要です — ドキュメントの基準がないと、複数ウィンドウでの同時開発はコードのスタイルがバラバラになったり、アーキテクチャの整合性が取れなくなったりする可能性があります。
- **ローカル実行によるデータの安全性：** OpenCodeは標準でローカルのOllamaモデルにつながるの、機密情報や公開前のビジネスロジックを含むプロジェクトに理想的な環境を提供します。Agent開発に必要なすべてのコードとドキュメントを完全にローカルで実行でき、サードパーティのAPIサービスに送信する必要がありません。

もともと苦手なこと

このやり方が一番苦手なのは、UIの細かい調整です。間隔、色合い、文字の見た目などの主観的な感覚は、Agentが実行できる具体的な指示に数値化するのが難しいです。Agentは数値（px、rem、色コード）しか理解できませんが、人間のインターフェースに対する印象は全体としてのものです。UIの細かい調整では、説明と議論に時間をかけても、最終的な満足度が見合わないことがあります。参考になるものがなくて調整が必要な場合、開発者が直接コードを直す方が、自然言語で見た目の印象を何度も説明してAgentに意図を推測させるよりも、効率も結果もよいです。

ワークフローと役割の変化

このやり方は、開発者の役割を「コードを書く」から「要件を決め、判断を下し、コードをレビューする」に変えます。開発者はもはや一行ずつコードを書くのではなく、要件を正確なドキュメントの仕様に変換し、大事な場面で技術的な判断を下し、Agentが生成したコードの品質と一貫性を確認することに集中します。

この変化は、基礎的な能力と業界での経験への要求を高めます。AI Agentは基本的にユーザーの指示に従うように作られており、ユーザーの理解が足りない部分を自分から見つけて指摘することはしません。ユーザーが特定の分野に経験がない場合、その考え方には標準的でない思い込みや、考えるべきことの抜けがあるかもしれませんが、Agentは自動的に補うことができません。効果的な対処法はプロンプトの工夫でAgentの役割を切り替えさせること（例：「業界の標準と比べて、今のやり方の問題点を指摘してください」）ですが、最終的な解決策の質は、やはりユーザーの考えの明確さとその分野の知識の深さに大きく依存します。

チームで使う場合の予想

このやり方をチーム（各メンバーがAgentを持つ）に広げた場合、いくつかの特徴が出てくると予想します。ドキュメントの検討段階では、複数の役割の調整（UI設計、フロントエンド、バックエンド、テスト）によってコミュニケーションの手間が大きく増え、Tech Leadが先に全体の枠組みを作り、各役割が自分の分野のドキュメントを詳しくしていく必要があります。開発とテストの段階では、ドキュメントファーストのやり方による素早い立ち上げの利点が活き、効率と質は各役割のドキュメントの詳しさと正確さに依存します。この予想は個人プロジェクトの経験にもとづいており、商業チームで完全に確かめたものではありません。

HR・管理層の視点

候補者が自分からAI Agentを使った開発を試みて、完全なワークフローを作り上げたことは、それだけで評価できる点です。考えられる質問：「Agentの助けがあるのに、なぜ開発期間にある程度の長さがあるのか」 — 理由は二つです：

- Agentの設定と調整には学ぶべきことがあり、結果は使う人のツールへの習熟度に左右される
- ドキュメントの細かさと出力の安定性の間には直接的なバランスがある — ドキュメントが詳しいほど（初期投資が大きいほど）、Agentの出力は安定する；速く立ち上げてあとで大きく調整するか、ゆっくり立ち上げてあとで少しだけ調整するかは、プロジェクトの納品速度と品質のどちらを重視するかによる

オープンソースと商用利用の境界

選んだ技術スタック（Next.js、React、Tailwind CSS、Framer Motion、Zodなど）は、すべて緩やかなライセンス（MIT、Apache 2.0など）を使っており、個人ポートフォリオや商用利用に関して問題はありません。OpenCode Agentもオープンソースのツールであり、企業が商用化のプロセスに組み込む場合は、ライセンスの条件を自分で確認する必要があります。

また、「AI Agentで開発する人の能力」について一般的な心配があります：Agentを使って開発する人は本当の能力が足りないと思う人もいます。しかし、実際はその逆です — Agentが人の指示に合わせる設計であるほど、使う人の業界経験や論理的な判断力に大きく依存します。Agentを効果的に使えば使えるほど、要件の分解、アーキテクチャ設計、品質管理といった基礎的な能力の高さがはっきり見えてきます。

反省とベストプラクティス

繰り返し直すよりもドキュメントを詳しく

最も大事な気づきは、一つの原則を確かめられたことです：**ドキュメントの段階に十分な時間を使って質の高い基準を作ることは、「すぐに始めて → 何度も直す」という繰り返しの方法よりも、長い目で見ると大きな効果がある**（具体的なデータは「主な課題 → 課題2」を参照）。これはよく言われる「素早い試作検証」とは反対の考え方です — 後者は要件がはっきりしないプロジェクトや自由に発想を広げるタイプのプロジェクトに向いていますが、今回のケースは開発者が明確な品質の期待を持っている場合、事前のドキュメントへの投資が効率の最適な答えであることを示しています。

次の似たようなプロジェクトでは、同じくらい詳しいドキュメントを保ちながら、もっと多くのAgent設定のパターン（スキルチェーンの連携、複数Agentの分担など）を試して、違う設定の戦略が出力の質に与える影響の範囲を探りたいと思います。プロセスの設計はできるだけ業界のソフトウェア工学の標準に沿っていますが、個人でやっているため商業チームの経験が足りず、標準の完全さと正確さは実際のチーム環境でもっと確かめられる必要があります — これは自分の能力の限界を示すと同時に、重点的に学ぶべき分野を示しています。

Agent開発の能力への理解

OpenCode Agentをゼロから設定する過程で、AI開発のツール全体に対する体系的な理解が得られました。skill（専門知識の提供）、MCP（Model Context Protocol）、AGENTS.md（プロジェクトの情報の定義）の3つの連携関係 — skillは分野の専門知識を提供し、AGENTS.mdはプロジェクト全体の制約を与え、MCPはツールの範囲を広げる — がAgent設定の中心的な骨組みを形作ります。

プロンプトエンジニアリングにおいては、「役割を切り替えるプロンプト」が効果的だと確かめられました：Agentに「業界の標準と比べて、今のやり方の問題点を指摘してください」や「この分野でよくあるけど、まだ考えられていない技術的なリスクを挙げてください」と頼むことで、Agentが人の指示に合わせる設計であることによる盲点の問題を部分的に減らせます。しかし、プロンプトの効果はユーザー自身のその分野の知識の深さに大きく依存します — ユーザーは潜在的な盲点を先に見つけられなければ、それを補うためのプロンプトを設計できません。AI支援開発の時代において、開発者の基礎的な能力への要求は下がるのではなく、上がっています。

ツール選びの実践的なアドバイス

Agentを使った開発のやり方を試してみたい開発者への実践的なアドバイス：予算が十分にあって、始めやすさを一番に考えるなら、Claude（Anthropic）が最も成熟したAgent製品と

して確実な選択です。低コストで始めて、ローカルで動かせることを重視するなら、OpenCodeが一番合っています — オープンソースなので、機密性の高いプロジェクトでもコードやドキュメントを外部のサービスに送信せずに、ローカルのOllamaモデルを使えます。OpenCodeからAgent開発を始めることで、無料の環境でAgentの設定、ドキュメントの連携、セッション管理など中心となる概念をしっかりと理解でき、他の有料製品への移行もスムーズになります。

結論と展望

中心となる発見

ドキュメントファースト + AI Agentの開発モデルは、個人プロジェクトにおいて明確な効果と投資に対する見返りを示しました。 中心となる仕組み：ソフトウェア工学の標準的なドキュメント体系（開発の全体の流れをカバーする20以上の番号付きドキュメント）をAgentの制約の枠組みと情報源として使い、人間の強み（要件定義、アーキテクチャの決定、品質レビュー）とAgentの強み（コード生成、ドキュメント同期、繰り返し作業の自動化）の協力のサイクルを作りました。最初の開発セッションで約80%の機能の骨組みを一度の生成で高い確率で達成し、ドキュメントの詳しさとAgentの出力の質に良い関係があることを確かめました。

もっと広い視点で見ると、AI支援開発の時代は開発者の能力の定義を変えています：ツールのハードルは下がる一方で、要件を分解する能力、アーキテクチャを判断する力、品質を管理する力への要求は高くなっています。Agentを効果的に使えば使えるほど、これらの基本的な分野における使い手の熟練度がはっきり見えてきます。

適用シナリオ

このやり方は以下のような場面に適しています：

- プロジェクトの要件にはっきりした期待を持っていて、素早い試作検証よりも納品の質を優先する開発者
- 規模が適切なプロジェクト（個人プロジェクトか小規模チーム）、ドキュメントを維持する手間が管理できる範囲
- ソフトウェア工学の知識があり、構造化された設計ドキュメントを作れる開発者

要件がはっきりせず自由な発想を重視するプロジェクトや、開発期間が極端に短いプロジェクトでは、ドキュメントの細かさを減らして立ち上げの速さを優先する方が現実的な選択かもしれません。

将来の方向性

いくつかの探求したい方向があります。一つ目は、Claudeや他の主要なAgent製品を比較テストに加えて、同じドキュメント体系の下で違うAgentの出力の質と開発の効率の違いを評価する計画です。二つ目は、このドキュメントファースト + Agentを使った進め方を他の個人プロジェクト（今考えているゲーム開発の計画など）に再利用し、方法論が分野をまたいで使えるかを確かめることです。最後に、ドキュメントの流れの自動化 — Agentやスクリプ

トツールを使ってドキュメントの一部を自動で生成したり同期できれば、全体の効率がもっと上がります。共通の目標：一つの実践を、標準化できて再利用可能な個人開発の方法論に少しずつ進化させることです。

キーワード

OpenCode Agent, ドキュメントファースト, Obsidian, Git, Vercel, ポートフォリオ, Next.js, AI開発, 人間とAIの協力, ゲーム風UI, Ollama

付録

付録A：ドキュメント体系完全リスト

ID	ドキュメント名	説明
0	ドキュメントインデックス	ドキュメント全体の地図と相互参照
1.1	個人ブランドとサイトビジョン	ポジショニング、目標、ブランドの雰囲気
1.2	機能一覧（MVP + 拡張）	機能の範囲と優先順位
2.1	サイト構造（サイトマップ）	ページ構造とルーティング設計
2.2	コンテンツ計画表	ページの内容とi18nキーの計画
3.1	スタイルガイド	デザイントークンとUI仕様
3.2	プロトタイプとワイヤーフレーム	レイアウトと情報の階層
3.3	インタラクションとモーション仕様	アニメーション仕様と操作の動き
4.1	技術スタックとアーキテクチャ	技術の選び方とアーキテクチャの決定
4.2	プロジェクトファイル構造	ディレクトリ構造とコンポーネントの役割
4.3	ビジュアルアセットと使用ガイドライン	画像とフォントリソースの仕様
5.1	コーディング規約	命名規則、TSルール、規約
5.2	開発タスク分解表	105タスクの分解とスケジューリング
6.1	デプロイガイド	Vercelでの手順とプロセス
6.2	環境設定表	Node.js、pnpmバージョン固定
7.1	コンテンツ更新ガイド	継続的な運用手順
7.2	SEOとアクセシビリティ	SEOメタデータとa11y仕様
—	VNゲーム風改造マニュアル	VN風の変更要件と実装の記録
—	VNゲーム風改造マニュアル第2弾	プロログシステムの仕様と実装の記録
—	完全技術仕様書と開発計画	総合的な技術仕様
—	AIを使った開発の概要とワークスペースルール	Agent設定と開発の進め方

ID	ドキュメント名	説明
—	更新記録	変更ログの要約（108行）
—	変更詳細	詳しい変更の記録（394行）
—	TODO開発チェックリスト	段階的なチェックリスト

付録B: OpenCode Agent設定の要点

opencode.json コア設定:

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

- external_directory: AgentにObsidianドキュメントの読み取り権限を与え、開発中に20以上の設計ドキュメントに直接アクセスできるようにする
- edit.ask: Obsidianドキュメントの変更には手動での確認が必要。Agentが確認なしで設計ドキュメントを変えるのを防ぐ
- instructions: frontend-designスキルを読み込み、Agentにフロントエンド設計の判断の指針を与える
- AGENTS.md: プロジェクトのルートにプロジェクトの情報を設定。フレームワークのバージョン、アーキテクチャのルール、コンポーネント仕様、ルーティングルールなどをカバー

付録C: 依存関係とバージョン詳細

カテゴリ	依存関係	バージョン
フレームワーク	next	16.2.9
フレームワーク	react / react-dom	19.2.4
国際化	next-intl	4.13.0
アニメーション	framer-motion	12.40.0
データ検証	zod	4.4.3
アイコン	lucide-react	1.18.0
CSSユーティリティ	clsx	2.1.1

カテゴリ	依存関係	バージョン
CSSユーティリティ	tailwind-merge	3.6.0
ビルドツール	tailwindcss	4.x
ビルドツール	@tailwindcss/postcss	4.x
型システム	typescript	5.x
リンター	eslint	9.x
リンター	eslint-config-next	16.2.9
パッケージ管理	pnpm	11.7.0

付録D: CI/CDパイプライン設定

```
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
```

パイプラインは lint → typecheck → build の順番を守ります。mainにプッシュすると自動で本番環境へのデプロイが始まり、PRを作成または更新するとプレビュー環境へのデプロイが始まります（Vercel GitHub Integrationが自動で管理）。

付録E: 開発環境仕様

項目	仕様
プロセッサ	11th Gen Intel Core i7-11700K @ 3.60 GHz
メモリ	16.0 GB RAM

項目	仕様
GPU	NVIDIA GeForce RTX 3060 12 GB
OS	Windows 11 64-bit
Node.js	>= 22.0.0
pnpm	11.7.0
ローカルAIモデル	Ollama（Qwen3等のオープンソースモデル）

参考資料

ツール/フレームワーク	バージョン	用途
OpenCode Agent	—	AIを使った開発Agent
Ollama	—	ローカルでLLMを動かす環境
Next.js	16.2.9	Reactのフレームワーク（App Router）
React	19.2.4	UIライブラリ
TypeScript	5.x	型システム
Tailwind CSS	4.x	CSSフレームワーク
Framer Motion	12.40.0	アニメーションライブラリ
next-intl	4.13.0	国際化のフレームワーク
Zod	4.4.3	データの検証
pnpm	11.7.0	パッケージ管理ツール
Vercel	—	デプロイプラットフォーム
GitHub Actions	—	CI/CD
Obsidian	—	ドキュメント管理
Git	—	バージョン管理

作者

KONG JUN SIENG

作成日

2026-07-07