1. Arquivo: Visão Geral do Projeto e Objetivos Estratégicos

Status: Documento Vivo (Versão Final Consolidada) Data: 30/11/2025

1.1. Propósito Central (Mission Statement)

"Construir não apenas um portfólio, mas um produto de software de alto desempenho que atue como prova viva da minha capacidade de engenharia. O site deve converter visitantes em oportunidades de entrevista através da demonstração prática de Performance (Core Web Vitals), Qualidade de Código e Inteligência Artificial Aplicada."

1.2. Definição do Público-Alvo e Comportamento

Precisamos atender a dois perfis com necessidades de tempo opostas:

- O "Scanner" (Recrutadores / RH):
    - Tempo no site: < 30 segundos.
    - Necessidade: Encontrar palavras-chave (skills), resumo de experiência e botão de contato/LinkedIn imediatamente.
    - Estratégia: "Above the fold" (dobra superior) extremamente clara e hierarquia visual óbvia.
- O "Auditor" (Líderes Técnicos / CTOs / Devs):
    - Tempo no site: > 2 minutos.
    - Necessidade: Verificar se você realmente sabe o que diz. Eles vão inspecionar o código-fonte, testar a performance no celular e ler os "Estudos de Caso" (post-mortems dos projetos).
    - Estratégia: Código aberto no GitHub (clean code), explicações arquiteturais profundas (curadoria de código) e features funcionais (como o Chatbot de IA).

1.3. Métricas de Sucesso (KPIs)

Como mediremos se o projeto valeu o esforço:

A. KPIs de Negócio (Conversão)

- Taxa de Contato: > 5% dos visitantes únicos clicam em "Email" ou "LinkedIn".
- Qualidade do Lead: Convites para entrevistas que citam especificamente projetos ou artigos lidos no site.
- Retenção: Tempo médio de sessão > 90 segundos (indica leitura dos estudos de caso).

B. KPIs Técnicos (Excelência de Engenharia)

- Lighthouse Score: 100/100 em Performance, Acessibilidade, Melhores Práticas e SEO.
- Core Web Vitals: Aprovação total (LCP < 2.5s, INP < 200ms, CLS < 0.1).
- Custo de Infraestrutura: Manter o projeto dentro do "Free Tier" da Vercel/AWS (demonstração de eficiência de arquitetura).

C. KPI de Diferenciação (O Fator "Uau")

- Engajamento com IA: Uso do Chatbot/Assistente por visitantes para fazer perguntas sobre o currículo (validando a integração do Vercel AI SDK).

1.4. Estratégia de Evolução (Pipeline de Atualização Ágil)

Para garantir que o site reflita o aprendizado contínuo sem se tornar um fardo de manutenção (Dívida Técnica):

- Regra dos 15 Minutos: A arquitetura ("Content as Data") deve permitir que a adição de um novo projeto ou nova habilidade leve menos de 15 minutos (tempo de criar o arquivo Markdown e fazer o commit).
- Princípio de Desacoplamento: Nenhuma informação de projeto ou skill deve ser "hardcoded" (chumbada) nos componentes visuais. Tudo deve vir de arquivos de dados (JSON/MDX) validados por schemas.

---

2. Arquivo: Stack Tecnológico e Arquitetura de Software

Status: Definido e Aprovado

Objetivo: Estabelecer as fundações técnicas para um produto de alta performance, fácil manutenção e focado na experiência do avaliador técnico.

2.1. A Stack "Gold Standard" (Decisões Rígidas)

Selecionamos ferramentas que são padrões de mercado para demonstrar proficiência imediata ("Market-Fit") e garantir os KPIs de performance (Core Web Vitals).

| Componente | Decisão | Justificativa Estratégica & Técnica |
|---|---|---|
| Framework | Next.js 14/15 (App Router) | Permite Renderização Híbrida (SSG para conteúdo estático, SSR para interações dinâmicas) essencial para SEO e velocidade. Facilita o uso de React Server Components (RSC). |
| Linguagem | TypeScript | Garante segurança de tipos e documentação automática do código. Demonstra maturidade em evitar bugs em tempo de execução. |
| Estilização | Tailwind CSS | Desenvolvimento acelerado com classes utilitárias. Garante consistência visual (Design System) e otimização automática de CSS (PurgeCSS) para produção. |
| Animação | Framer Motion | Biblioteca padrão para animações declarativas em React. Permite interações complexas (layout transitions) sem sacrificar a performance do thread principal. |
| Validação de Dados | Zod | Garante a integridade do conteúdo ("Content as Data"). Se um projeto faltar um campo obrigatório no JSON/Frontmatter, o build falha propositalmente (Fail Fast). |
| Testes | Vitest + Playwright | Vitest: Para testes unitários rápidos. Playwright: Para testes E2E que simulam o navegador do recrutador (cliques, navegação, responsividade). |

| Componente | Decisão | Justificativa Estratégica & Técnica |
|---|---|---|
| Infraestrutura | Vercel | Hospedagem Zero-Config otimizada para Next.js. Garante deploy automático via Git, HTTPS global e Edge Caching. |

2.2. Estratégia de Integração com GitHub: "O Curador de Código"

O site não hospedará a base de código completa dos projetos. Ele atuará como uma camada de inteligência e navegação sobre os seus repositórios no GitHub.

A. Conceito de "Deep Linking" (Links Profundos)

Ao invés de replicar arquivos, os "Estudos de Caso" no site conterão links diretos para linhas específicas do GitHub que demonstram a solução.

- *Mecanismo:* O link deve levar para o "Permalink" do commit específico (para garantir que o link não quebre se o arquivo mudar no futuro).
- *Exemplo de UX:* "Veja como resolvi a race condition na linha 42 do arquivo api/auth.ts."

B. "Golden Snippets" (Trechos de Ouro)

Exibiremos código no site apenas quando for necessário explicar um raciocínio complexo.

- Regra: Máximo de 15 linhas por snippet.
- Objetivo: Ilustrar algoritmos, hooks customizados ou configurações de infraestrutura. Não mostrar boilerplate.

C. Diagramas sobre Código

Prioridade para diagramas (Mermaid.js ou imagens exportadas do Excalidraw) para explicar fluxos de dados e arquitetura de microsserviços. Isso demonstra visão sistêmica.

2.3. Arquitetura para Uso de IA (O Diferencial Funcional)

A integração de IA não será apenas "geração de texto", mas uma feature funcional do site (Chatbot/Assistente).

- Vercel AI SDK: Abstração padrão para conectar o front-end aos modelos de linguagem (LLMs).
- Edge Functions (Serverless): O endpoint da API do chat rodará na Edge (Vercel Edge Functions) para latência mínima.
- Segurança: As chaves de API (Gemini/OpenAI) ficam estritamente no servidor (Variáveis de Ambiente), nunca expostas ao cliente.
- Fallback: Se a cota da API acabar, a interface deve degradar graciosamente (ex: esconder o chat ou mostrar uma mensagem amigável), mantendo o resto do site funcional.

2.4. Estrutura de Deploy e CI/CD (Pipeline)

O fluxo de trabalho deve ser profissional desde o primeiro commit.

1. Commit (Local): Husky (git hooks) roda o *linter* e verifica tipos antes de permitir o commit.
2. Push (GitHub): Aciona o Vercel Preview.
3. Preview (Ambiente de Teste): A Vercel gera uma URL única para aquela branch.
4. Testes Automatizados (Opcional/Ideal): GitHub Actions roda a suíte do Playwright contra a URL de Preview.
5. Merge para Main: Aciona o Deploy de Produção instantâneo.

3. Arquivo: Estrutura de Dados e Componentização (Component Library)

Objetivo: Definir a organização rígida dos dados e a arquitetura de pastas para evitar dívida técnica e garantir que o conteúdo seja desacoplado da interface visual.

3.1. Arquitetura de Pastas e Organização do Projeto [ATUALIZADO]

Para suportar a stack Next.js (App Router) e a estratégia de "Content as Data", o projeto seguirá esta estrutura de diretórios estrita:

Plaintext

src/

```
├── app/                # Next.js App Router (Páginas e Rotas)
│   ├── layout.tsx      # MainLayout (SEO Global, Header, Footer)
│   ├── page.tsx        # Página Inicial (Home)
│   ├── projetos/       # Rotas Dinâmicas de Projetos
│   │   └── [slug]/     # Página de Detalhes (renderiza o MDX)
│   └── api/            # Rotas de Backend (Serverless)
│       └── chat/       # Endpoint para o Micro-Serviço de IA
├── components/         # Biblioteca de Componentes (Ver item 3.3)
│   ├── ui/             # Componentes Atômicos (Botões, Inputs)
│   ├── sections/       # Seções de Layout (Hero, Sobre, Skills)
│   └── project/        # Componentes Específicos de Projeto
├── content/            # Repositório de Conteúdo ("Content as Data")
│   ├── projects/       # Arquivos .mdx (Um por projeto)
│   └── profile.json    # Dados estáticos do "Sobre Mim"
├── lib/                # Lógica Utilitária
│   ├── utils.ts        # Helpers (cn, formatters)
│   └── schemas.ts      # Validações Zod (Schema Rígido)
└── public/             # Assets Estáticos (Imagens, Fontes)
```

3.2. Estrutura de Conteúdo (Dados Imutáveis e Validação)

O site é um repositório de conteúdo estático tratado como dados. Utilizaremos MDX para o conteúdo rico e Zod para garantir que nenhum dado obrigatório esteja faltando.

Schema Rígido para Projetos (src/lib/schemas.ts): Todo arquivo de projeto deve conter obrigatoriamente os seguintes campos no frontmatter, validados via Zod:

- id/slug: string (Identificador único para a URL)

- title: string (Nome do Projeto)
- tech_stack: array<string> (Lista de tecnologias usadas, ex: ["Next.js", "AWS"])
- description_short: string (Headline de impacto para o card)
- problem: string (A "dor" que o projeto resolveu)
- solution: string (A abordagem técnica utilizada)
- result_quantified: string (O impacto de negócio/KPI)
- github_link: string (URL para o repositório)
- live_demo_link: string (URL para o projeto em produção)
- featured: boolean (Define se aparece no topo da Home)

Habilidades (Skills): Lista estruturada em JSON ou objeto TypeScript constante para gerar a "Skills Matrix": { category: string, skills: string[] }.

3.3. Biblioteca de Componentes (Design System)

Os componentes devem ser construídos isoladamente para garantir consistência visual em todas as páginas.

A. Componentes Atômicos (src/components/ui): Elementos indivisíveis e reutilizáveis:

- <Button />: Variantes primária (CTA), secundária e ghost.
- <TechPill />: Badges para exibir as tecnologias usadas.
- <Icon />: Wrapper para ícones SVG (Lucide React ou similar).

B. Componentes Modulares (src/components/project, src/components/sections): Blocos de construção da interface:

- <ProjectCard />: Recebe o objeto project validado e renderiza o card resumo.
- <Navbar />: Navegação responsiva com suporte a Dark Mode.
- <Footer />: Links sociais e copyright.
- <JsonLd />: Componente invisível para injetar SEO estruturado nas páginas.

C. Layouts Globais:

- MainLayout: Define o grid principal, fontes e SEO base.
- SectionContainer: Wrapper para garantir padding horizontal e largura máxima consistente (container).

3.4. Configuração de Tema (Design Tokens)

Utilizar o arquivo de configuração do Tailwind CSS (tailwind.config.ts) como a única fonte da verdade para:

- Cores: Definir variáveis semânticas (ex: primary, background, surface) em vez de valores hexadecimais diretos.
- Fontes: Configurar a família tipográfica principal e secundária.
- Espaçamento: Manter a escala padrão do Tailwind para consistência.

---

4. Arquivo: Planejamento de Design e UX (Design System)

Status: Atualizado para Suportar Curadoria de Código Objetivo: Criar uma interface que transmita "Senioridade" e "Organização" instantaneamente, facilitando a vida dos perfis "Scanner" e "Auditor".

4.1. Identidade Visual (Thematic Definition)

O tema deve evocar o ambiente de desenvolvimento moderno (IDE-inspired).

- Modo Padrão: Dark Mode (Fundo escuro reduz o cansaço visual e remete a editores de código como VS Code).
- Paleta de Cores:
    - Background: Slate-950 ou Zinc-950 (Não use preto puro #000, use um cinza profundo para suavidade).
    - Surface (Cards/Paineis): Slate-900 com bordas sutis Slate-800.
    - Primary (Ação/Links): Um tom vibrante, mas legível. Sugestão: Violet-500 (Moderno/AI) ou Emerald-500 (Sucesso/Performance) .

- Syntax Colors: Baseado no tema "Dracula" ou "One Dark Pro" para os snippets de código (garante familiaridade para quem lê).

4.2. Componentes de UX Específicos (Novos)

Para suportar a estratégia de "Deep Linking" e "Golden Snippets" definida no Arquivo 2.

A. O Componente "Deep Link" (O Botão de Prova) Visualmente distinto de um link comum. Deve parecer um "ticket" ou referência técnica.

- Visual: Ícone do GitHub pequeno + Nome do Arquivo + Intervalo de Linhas.
- Exemplo: [GitHub Icon] src/hooks/useAuth.ts : Lines 45-82
- Interação: Hover revela uma tooltip: "Ver código fonte no GitHub".

B. O Container de Diagramas Para diferenciar diagramas de screenshots comuns.

- Estilo: Fundo sutilmente quadriculado (dot pattern) para remeter a quadros brancos digitais (Excalidraw/Miro).
- Legenda: Obrigatória e descritiva na parte inferior.

C. O "Golden Snippet" (Syntax Highlighting)

- Ferramenta: Usar Shiki ou Prism rodando no servidor (para não pesar o JS no cliente).
- UI: Deve ter uma barra de título com o nome da linguagem e um botão "Copy".
- Restrição: Altura máxima definida (com scroll ou gradiente de "leia mais") para não dominar a página.

4.3. Tipografia e Hierarquia

- Fontes de Título (Headings): *Inter* ou *Geist Sans* (Limpa, moderna, suíça).
- Fontes de Código (Monospace): *JetBrains Mono* ou *Fira Code*. Usar não apenas em blocos de código, mas para destacar Tecnologias e KPIs no texto corrido.

4.4. Animações Funcionais (Framer Motion)

Nada de "balançar" elementos sem motivo.

- Transição de Página: Sutil (fade-in) para não cansar.
- Micro-interações: Feedback tátil ao clicar nos botões de "Copiar Código" ou ao passar o mouse nos Cards de Projeto.

5. Arquivo: SEO Técnico e Estratégia de Lançamento (Next.js Special)

Status: Atualizado para Next.js 14/15 Metadata API Objetivo: Garantir que o site seja encontrável e, principalmente, que os links compartilhados no LinkedIn/WhatsApp sejam irresistíveis (Social SEO).

5.1. SEO Técnico (A Nova Abordagem Next.js)

Esqueça as tags <head> manuais. Usaremos a Metadata API do Next.js para controle total e dinâmico.

A. Metadados Dinâmicos por Projeto Cada página de projeto (src/app/projetos/[slug]/page.tsx) deve exportar uma função generateMetadata:

- Título: Nome do Projeto | Seu Nome
- Descrição: A "Short Description" definida no frontmatter (focada em resultado).
- Keywords: O array de tech_stack do projeto é injetado automaticamente aqui.

B. Open Graph Dinâmico (og-image) - O Diferencial Usaremos a biblioteca @vercel/og para gerar imagens de compartilhamento *on-the-fly*.

- O que é: Quando você colar o link do seu projeto no LinkedIn, não aparecerá uma foto genérica.
- O que aparece: Uma imagem gerada via código contendo:
    - O Título do Projeto.
    - Os 3 principais ícones da Stack.
    - O KPI principal (ex: "-40% Latência").

- Impacto: Aumenta drasticamente o CTR (Click-Through Rate) de recrutadores.

5.2. Core Web Vitals (O KPI de Performance)

O SEO técnico depende da performance.

- Imagens: Uso obrigatório do componente <Image /> do Next.js para conversão automática para WebP/AVIF e prevenção de Layout Shift (CLS).
- Fontes: Uso de next/font para zero layout shift no carregamento das fontes.

5.3. Estratégia de Lançamento e Monitoramento

Fase 1: Validação "Silenciosa" (Pré-Launch)

1. Lighthouse CI: Configurar no GitHub Actions para que *todo* Pull Request falhe se a pontuação de performance cair abaixo de 90 .
2. Teste de Acessibilidade: Garantir navegação por teclado (Tab) e leitores de tela (ARIA labels), crucial para empresas que valorizam inclusão.

Fase 2: Divulgação Estratégica ("Build in Public") Ao invés de apenas postar "Site novo no ar", crie conteúdo sobre *como* você o fez.

1. Post no LinkedIn: "Como usei Next.js e Vercel AI SDK para criar um portfólio que se explica sozinho". (Isso atrai recrutadores técnicos).
2. Repo no GitHub: O README.md do portfólio deve ser impecável, explicando a arquitetura (Deep Links, Content as Data) para quem cair lá direto.

Fase 3: Manutenção (Analytics Privado)

- Ferramenta: Usar Vercel Web Analytics (focado em privacidade e performance) em vez de Google Analytics pesado.
- Objetivo: Monitorar quais projetos estão sendo mais clicados para reordenar a Home page (colocar os vencedores no topo).

## 6. Arquivo: Estrutura de Conteúdo e Narrativa dos Projetos [REFINADO]

Status: Atualizado para estratégia "Curadoria de Código" Objetivo: Transformar cada página de projeto em um "tour guiado" pela sua engenharia, guiando o avaliador técnico diretamente para os pontos de interesse no GitHub.

6.1. O Card Resumo (A Isca na Página Inicial)

Este elemento deve funcionar como um "Elevator Pitch" visual. Ele não deve contar a história inteira, mas vender o resultado para forçar o clique.

- Fonte de Dados: Frontmatter do arquivo .mdx (validado pelo Zod).
- Elementos Visuais:
    1. Tag de Impacto (Novo): Uma badge de topo que destaca o principal feito (ex: "🚀 -40% Latência" ou "💰 R$ 10k Economia/mês").
    2. Tech Stack Visual: Ícones das 3 tecnologias principais.
    3. Título: Nome do projeto.
    4. Resumo de Solução: *Não* diga o que o app faz (ex: "Lista de tarefas"). Diga o problema técnico resolvido (ex: "Sincronização Offline-First com Conflict Resolution").
    5. Ação: Botão "Ler Estudo de Caso".

6.2. A Página de Detalhes (O Estudo de Caso / "The Tour")

Aqui aplicamos a estratégia de Curadoria. A narrativa deve seguir a ordem lógica de avaliação de um CTO.

A. O Cabeçalho (O Contexto de Negócio) Antes de falar de código, prove que você entende o negócio.

- O Desafio: Qual era a dor real? (Ex: "O processo de checkout demorava 4s, causando 20% de abandono").
- A Meta: O que era sucesso? (Ex: "Reduzir para < 1s").
- Resultado Quantificado (KPI): O número final em destaque grande.

B. A Arquitetura (Visão Sistêmica) Substituímos screenshots genéricos por Diagramas.

- O Que Mostrar: Fluxo de dados, interação entre microsserviços ou estrutura do banco de dados.
- Formato: Imagem exportada do Excalidraw ou Mermaid.js renderizado.
- Legenda: "Diagrama de fluxo de autenticação OAuth 2.0".

C. "Deep Dive" Técnico (A Curadoria de Código) Esta é a seção que substitui os grandes blocos de código. Crie parágrafos focados em desafios específicos seguidos de provas.

- Estrutura do Bloco:
    1. O Problema Específico: "Precisávamos garantir que transações financeiras fossem atômicas mesmo em falhas de rede."
    2. A Solução Lógica: "Implementei o padrão Saga com compensação de eventos."
    3. A Prova (Deep Link): Botão estilizado: [🔗 Ver implementação da Saga no GitHub (Linhas 45-80)]
    4. O "Golden Snippet" (Opcional): Apenas se houver um trecho de código muito elegante ou complexo (máx 15 linhas) que mereça ser lido na hora. Use highlight de sintaxe (Prism/Shiki).

D. Lições e Melhorias (Maturidade)

- O que eu faria diferente: "Hoje, eu substituiria o Redux por Zustand para simplificar o boilerplate..." (Isso mostra que você continua evoluindo).

6.3. Template de Escrita (Checklist para Novos Projetos)

Para cumprir a regra dos 15 minutos de atualização (Arquivo 1), use este template mental ao escrever o .mdx:

1. [ ] Headline: Qual o número mais impressionante desse projeto?
2. [ ] Diagrama: Tenho um desenho da arquitetura?
3. [ ] Link 1: Onde está a lógica mais difícil no GitHub? (Copiar Permalink).
4. [ ] Link 2: Onde está a configuração de infra/banco? (Copiar Permalink).

5. [ ] Tech Stack: Quais as 5 tecnologias para o array?

7. Arquivo: Estrutura da Seção "Sobre Mim" e Branding Pessoal [REFINADO]

Status: Atualizado para "Data-Driven Storytelling" Objetivo: Humanizar o "Produto de Engenharia" e servir como ponto de entrada para a interação com a IA.

7.1. Estrutura de Dados (A Fonte da Verdade)

Seguindo nossa regra de "Content as Data", nada aqui será hardcoded. As informações virão de um arquivo profile.json ou about.mdx.

A. O "Elevator Pitch" Dinâmico Texto curto para a Hero Section ou Sidebar.

- Scanner (Recrutador): Vê o título: "Engenheiro de Software focado em Performance e IA".
- Auditor (Técnico): Vê a stack logo abaixo.

B. A Jornada (Timeline Interativa) Em vez de grandes blocos de texto, usaremos uma estrutura de dados de "Marcos de Carreira" (career_milestones).

- Estrutura do JSON:
    - date: "2023"
    - title: "Liderança Técnica no Projeto X"
    - description: "Onde aprendi a gerenciar conflitos de merge e arquitetura de microsserviços."
    - icon: "Briefcase" ou "GraduationCap"

7.2. Componentes de UI (A Experiência Visual)

A. Avatar com "Presence Indicators"

- Visual: Sua foto profissional (bem iluminada, fundo neutro).
- Tech Twist: Um anel de status pulsante (CSS/Framer Motion) que indica "Disponível para projetos" ou "Open to Work".

B. A Matrix de Habilidades (Skills Matrix 2.0)

- Adeus Barras de Porcentagem: Como você bem notou no original, "80% de React" é subjetivo e perigoso.
- Nova Abordagem: Tags categorizadas renderizadas a partir do seu JSON.
    - Categorias: "Linguagens", "Frameworks", "DevOps", "Ferramentas".
    - Diferencial: Ao passar o mouse em uma skill (ex: "Next.js"), o site realça/brilha os cartões de projetos onde essa skill foi usada (conexão cruzada de dados).

C. O "Hook" para a IA (Call to Action) Esta seção é o lugar perfeito para introduzir o Chatbot.

- Texto: "Quer saber mais detalhes sobre minha experiência ou hobbies? Pergunte para minha IA."
- Botão: "💬 Conversar com o Assistente" (Abre o modal do chat alimentado pelo Vercel AI SDK).

7.3. Conteúdo da Narrativa (O Texto em Si)

A. Profissional vs. Pessoal (Equilíbrio Cultural)

- Foco Profissional (80%): Resolvendo problemas complexos, paixão por "Clean Code" e arquitetura escalável.
- Foco Pessoal (20%): Hobbies que demonstram *soft skills*.
    - *Exemplo:* "Jogador de Xadrez (Pensamento Estratégico)" ou "Maratonista (Resiliência e Disciplina)".

B. Testemunhos (Prova Social)

- Formato: Cards simples com citações curtas extraídas do LinkedIn.
- Curadoria: Escolha frases que elogiem não só seu código, mas sua comunicação e entrega.

---

Resumo das Mudanças

1. Remoção de Subjetividade: Substituímos "barras de progresso" por uma conexão inteligente entre Skills e Projetos.
2. Integração com IA: Transformamos o "Sobre Mim" no gatilho principal para o uso do Chatbot.
3. Timeline: Transformamos "textão" em um componente visual de linha do tempo, mais fácil de escanear.

**8. Arquivo: Fluxo de Trabalho de Desenvolvimento (CI/CD) e Plano de Qualidade**

**Status: Atualizado para Stack Moderno (Vitest/Playwright) Objetivo: Garantir que o portfólio esteja "sempre pronto para produção" e que erros sejam capturados automaticamente antes do deploy.**

**8.1. Estratégia de Versionamento (Git Workflow Profissional)**

**Adotaremos uma postura rígida de commits para demonstrar organização no histórico do GitHub.**

- **Padrão de Branchs:**
    - **main: Código em produção (Protegido).**
    - **feature/nome-da-feature: Para novas funcionalidades (ex: feature/dark-mode).**
    - **content/nome-do-projeto: Para adição de novos estudos de caso (MDX).**
- **Conventional Commits: Usar prefixos semânticos nas mensagens de commit. Isso gera changelogs automáticos e mostra disciplina.**
    - **feat: Nova funcionalidade.**
    - **fix: Correção de bug.**
    - **content: Adição ou edição de projetos/textos.**
    - **chore: Configuração de ferramentas/deps.**

**8.2. O Pipeline de Integração Contínua (CI - The Quality Gate)**

**Sempre que você abrir um Pull Request (PR), o GitHub Actions (ou Vercel CI) deve rodar 3 verificações obrigatórias. Se uma falhar, o botão de "Merge" fica bloqueado.**

1. **Type Check & Lint:**
    - **tsc --noEmit: Verifica se não há erros de tipagem TypeScript.**
    - **eslint: Garante que o estilo do código segue o padrão.**
2. **Validação de Conteúdo (Zod) [CRÍTICO]:**
    - **Um script que lê todos os arquivos .mdx e .json e valida contra o Schema do Zod (definido no Arq 3).**
    - ***Objetivo:* Impedir que um projeto vá para o ar sem "Resumo" ou sem "Tech Stack".**
3. **Testes Automatizados: Execução da suíte de testes (ver 8.3).**

**8.3. Estratégia de Testes (A Pirâmide de Testes Next.js)**

**Substituímos Jest/Cypress por ferramentas mais leves e nativas para o ecossistema Vite/Next.js.**

**A. Testes Unitários (Vitest) Focados em lógica isolada e utilitários.**

- **O que testar:**
    - **Formatadores de data.**
    - **Lógica de filtragem de projetos.**
    - **Validação dos Schemas Zod.**
- **Por que Vitest? Roda nativamente com a configuração do projeto, sendo muito mais rápido que o Jest.**

**B. Testes End-to-End (Playwright) Simula o "Recrutador" navegando no site.**

- **Cenários Críticos:**
    1. ***Fluxo de Contato:* Preencher o formulário e verificar se a API foi chamada (mockada).**
    2. ***Navegação:* Clicar em um Projeto na Home e verificar se a página de detalhes carregou.**
    3. ***Responsividade:* Verificar se o Menu Hambúrguer abre no Mobile.**

4. ***Deep Links:*** **Verificar se os links externos para o GitHub não retornam 404.**

**C. Testes de IA (Mocking)**

- **Regra: Nunca gastar tokens reais da API (OpenAI/Gemini) nos testes automatizados.**
- **Estratégia: Mockar a resposta do Vercel AI SDK nos testes para garantir que a UI do Chatbot (loading state, mensagem de erro, renderização de markdown) funcione sem chamar a IA de verdade.**

**8.4. Monitoramento e Deploy (CD)**

**A infraestrutura pós-merge.**

- **Deploy Automático (Vercel):**
  - **Push na main -> Deploy de Produção.**
  - **Push em feature/* -> Deploy de Preview (Gera uma URL única para você testar no celular antes de aprovar).**
- **Auditoria de Performance (Lighthouse CI):**
  - **Configurar para falhar o deploy se o Performance Score cair abaixo de 95/100. Isso garante que você nunca introduza uma imagem pesada ou script bloqueante acidentalmente.**
- **Analytics (Vercel Web Analytics):**
  - **Ativar para medir visualizações reais (Privacy-friendly, sem cookies de rastreamento intrusivos).**