Code Guardian

Escopo Técnico Detalhado do MVP

com Stack e Estrutura de Implementação

Introdução

Este documento descreve de forma extremamente detalhada o escopo do MVP do projeto Code Guardian, incluindo a stack tecnológica recomendada e a estrutura completa de implementação. O objetivo é eliminar ambiguidades técnicas, reduzir decisões ad-hoc durante o desenvolvimento e servir como referência única de implementação. O MVP foi intencionalmente desenhado para ser poderoso, porém limitado, permitindo validação rápida sem acúmulo prematuro de complexidade.

1. Problema que o MVP Resolve

- Desenvolvedores não entendem o impacto real de mudanças locais no código.
- Arquiteturas degradam silenciosamente até se tornarem frágeis.
- Regras críticas de negócio não são protegidas em runtime.
- Ferramentas atuais focam em sintaxe, não em decisões.

Importante: O MVP NÃO tenta corrigir código, sugerir refactors automáticos ou substituir revisão humana. Ele existe para alertar, conscientizar e proteger.

2. Visão de Produto do MVP

O Code Guardian MVP se posiciona como um copiloto técnico que atua em dois momentos distintos: durante o desenvolvimento (DX) e durante a execução (runtime). Essas duas camadas compartilham conceitos, linguagem e modelo mental.

- **DX:** Antecipar impacto e risco antes do código rodar.
- **Runtime:** Garantir que invariantes críticas não sejam violadas.

3. Arquitetura de Alto Nível

O MVP será implementado como um monorepo contendo três pacotes desacoplados. Nenhum pacote pode depender diretamente de APIs específicas dos outros.

Pacote	Responsabilidade
@code-guardian/core	Núcleo analítico e modelo de risco
code-guardian-vscode	Extensão VS Code focada em impacto e carga cognitiva
@code-guardian/runtime	Biblioteca Node.js para proteção de fluxos

4. Escopo Técnico Detalhado — @code-guardian/core

4.1 Responsabilidades

- Construir grafo de dependências a partir de AST TypeScript
- Calcular métricas estruturais (fan-in, fan-out, profundidade)
- Gerar modelo determinístico de risco por arquivo
- Expor API pura e testável

4.2 Modelo de Risco (MVP)

Nível	Características
Baixo Risco	Baixo fan-in, pequeno tamanho, poucas dependências
Médio Risco	Crescimento moderado, múltiplos dependentes
Alto Risco	Arquivo central, muitos dependentes, alta complexidade

4.3 Fora de Escopo

- Análise semântica profunda de regras de negócio
- Suporte a linguagens além de TypeScript

5. Escopo Técnico Detalhado — code-guardian-vscode

5.1 Funcionalidades Incluídas

- **Code Impact Lens:** exibição de dependências diretas e indiretas
- **Indicadores visuais de risco** no editor
- **Cognitive Load Meter** baseado em heurísticas simples
- **Mensagens educativas,** não prescritivas

5.2 Experiência do Usuário

- Nenhuma ação bloqueante
- Warnings apenas informativos
- Zero modificação automática de código

6. Escopo Técnico Detalhado — @code-guardian/runtime

6.1 Objetivo

Garantir que invariantes críticas definidas pelo desenvolvedor não sejam violadas durante a execução da aplicação.

6.2 API do MVP

- `guard.require(conditionName)`
- `guard.ensure(expression)`
- Erros explícitos e previsíveis

6.3 Limitações

- Não intercepta chamadas automaticamente
- Não mantém estado global oculto

7. Critérios de Aceitação do MVP

- Projeto real consegue visualizar impacto de mudanças
- Runtime impede ao menos um fluxo inválido real
- Core reutilizado por DX e Runtime sem acoplamento

8. Roadmap Pós-MVP (Referencial)

- Integração automática DX → Runtime
- Análise histórica e architectural drift
- CI / PR automation

9. Stack Tecnológica Recomendada

A stack escolhida prioriza performance, experiência do desenvolvedor (DX) e alinhamento com o ecossistema TypeScript. Todas as escolhas foram feitas para maximizar velocidade de desenvolvimento sem comprometer a qualidade ou escalabilidade futura.

9.1 Visão Geral da Stack

Categoria	Ferramenta	Justificativa
Linguagem	TypeScript	Análise nativa de código TS
Runtime	Node.js 18+	ESM nativo, estabilidade
Gerenciador	pnpm	Performance superior, hard links
Build System	Turborepo	Cache inteligente, paralelização
Bundler	tsup / esbuild	Velocidade extrema, zero config
Testing	Vitest	10-20x mais rápido que Jest
Linter	ESLint 9+	Análise estática moderna
Formatter	Prettier	Formatação consistente
CI/CD	GitHub Actions	Integração nativa
Versionamento	Changesets	Semver para monorepos

9.2 Análise de Código (@code-guardian/core)

Ferramenta	Propósito
TypeScript Compiler API	Análise oficial de AST TypeScript
ts-morph (alternativa)	Wrapper amigável sobre Compiler API
graphlib	Construção e análise de grafos
@typescript-eslint/parser	Parsing adicional para análise estática

9.3 VS Code Extension

Importante: Para o MVP, a extensão não necessita de frameworks como React ou Svelte. As APIs nativas do VS Code (tree views, status bar, decorators) são suficientes. Frameworks devem ser considerados apenas para visualizações complexas (grafos interativos, dashboards).

• VS Code Extension API - infraestrutura base da extensão
• vscode-languageserver - features avançadas LSP
• esbuild - bundle ultra-rápido

10. Estrutura Detalhada do Projeto

A estrutura do monorepo foi desenhada para maximizar o desacoplamento entre pacotes, facilitar reutilização de código e permitir desenvolvimento independente de cada componente.

10.1 Estrutura de Diretórios

```
code-guardian/
■
■■■ packages/
■ ■
■ ■■■ core/ # Núcleo analítico
■ ■ ■■■ src/
■ ■ ■ ■■■ ast/ # Análise de AST
■ ■ ■ ■ ■■■ parser.ts
■ ■ ■ ■ ■■■ visitor.ts
■ ■ ■ ■■■ graph/ # Construção de grafos
■ ■ ■ ■ ■■■ builder.ts
■ ■ ■ ■ ■■■ analyzer.ts
■ ■ ■ ■■■ metrics/ # Cálculo de métricas
■ ■ ■ ■ ■■■ fanin.ts
■ ■ ■ ■ ■■■ fanout.ts
■ ■ ■ ■ ■■■ complexity.ts
■ ■ ■ ■■■ risk/ # Modelo de risco
■ ■ ■ ■ ■■■ calculator.ts
■ ■ ■ ■ ■■■ classifier.ts
■ ■ ■ ■■■ index.ts # Public API
■ ■ ■■■ tests/
■ ■ ■■■ package.json
■ ■ ■■■ tsconfig.json
■ ■
■ ■■■ vscode/ # Extensão VS Code
■ ■ ■■■ src/
■ ■ ■ ■■■ extension.ts # Entry point
■ ■ ■ ■■■ providers/ # LSP providers
■ ■ ■ ■ ■■■ impact-lens.ts
■ ■ ■ ■ ■■■ decorators.ts
■ ■ ■ ■■■ views/ # Tree views
■ ■ ■ ■ ■■■ dependencies.ts
■ ■ ■ ■■■ commands/ # Comandos
■ ■ ■■■ package.json
■ ■ ■■■ tsconfig.json
■ ■ ■■■ .vscodeignore
■ ■
■ ■■■ runtime/ # Runtime guards
■ ■■■ src/
■ ■ ■■■ guard.ts # API principal
■ ■ ■■■ errors.ts # Error handling
■ ■ ■■■ types.ts # Type definitions
■ ■ ■■■ index.ts
■ ■■■ tests/
■ ■■■ package.json
■ ■■■ tsconfig.json
```

```
■
■■■ .github/workflows/ # CI/CD
■ ■■■ ci.yml # Tests + lint
■ ■■■ release.yml # Releases automáticos
■
■■■ pnpm-workspace.yaml # Config do workspace
■■■ turbo.json # Build pipeline
■■■ tsconfig.base.json # TS config compartilhado
■■■ .prettierrc # Formatação
■■■ .eslintrc.js # Linting
■■■ package.json # Root package
```

10.2 Configurações Essenciais

pnpm-workspace.yaml

```
packages:
- 'packages/*'
```

turbo.json

```
{
"pipeline": {
"build": {
"dependsOn": ["^build"],
"outputs": ["dist/**"]
},
"test": {
"dependsOn": ["build"]
},
"lint": {}
}
}
```

10.3 Fluxo de Dependências

O design garante que não existam dependências circulares e que cada pacote possa ser desenvolvido e testado independentemente:

Pacote	Depende de	Usado por
@code-guardian/core	(nenhum)	vscode, (futuro) runtime
code-guardian-vscode	@code-guardian/core	(nenhum)
@code-guardian/runtime	(independente)	(aplicações do usuário)

11. Justificativas Técnicas das Escolhas

11.1 Por que pnpm + Turborepo?

Ferramenta	Benefícios	Impacto
pnpm	Hard links, dedupe automático	Instalação 2x mais rápida, economia de disco
Turborepo	Cache local/remoto, paralelização	Redução de 85% no tempo de build

11.2 Por que Vitest ao invés de Jest?

Aspecto	Jest	Vitest
Performance	Baseline	10-20x mais rápido
ESM	Requer transformers	Suporte nativo
Configuração	Complexa	Zero config
HMR	Não disponível	Re-run instantâneo

11.3 Ferramentas a Evitar

As seguintes ferramentas, embora populares, tornaram-se obsoletas para projetos modernos:

Ferramenta	Problema	Alternativa
Webpack	Lento (10-50x)	esbuild / Vite
Jest	ESM complexo	Vitest
Lerna	Obsoleto	pnpm workspaces
Babel	Layer extra	TypeScript nativo

12. Diretrizes de Implementação

12.1 Ordem de Desenvolvimento Recomendada

Fase	Escopo	Entregável
1	Setup do monorepo	pnpm + Turborepo + configs
2	Core - AST parsing	Análise de dependências
3	Core - Modelo de risco	Classificação de arquivos
4	VSCode - Integração	Code Impact Lens
5	VSCode - Indicadores	Visual feedback
6	Runtime - API	guard.require/ensure
7	Validação	Teste em projeto real

12.2 Princípios de Qualidade

- **Test First:** Cada feature deve ter testes antes da implementação
- **API pública mínima:** Exportar apenas o necessário
- **Zero dependências ocultas:** Declaração explícita em cada pacote
- **Documentação inline:** JSDoc em todas as funções públicas
- **Versionamento semântico:** Usar Changesets para releases

12.3 Performance Targets

Componente	Métrica	Target
Core analysis	Projeto 1000 arquivos	< 5 segundos
VS Code extension	Ativação	< 500ms
Runtime guards	Overhead por guard	< 1ms
Build completo	Com Turborepo cache	< 30 segundos

13. Considerações Finais

Este documento estabelece as bases técnicas completas para a implementação do Code Guardian MVP. A stack escolhida prioriza performance, experiência do desenvolvedor e manutenibilidade futura, enquanto a estrutura do projeto garante desacoplamento e escalabilidade.

13.1 Benefícios da Arquitetura Proposta

- **Desenvolvimento independente:** Cada pacote pode evoluir sem impactar os outros
- **Performance otimizada:** Stack moderna garante builds e testes rápidos
- **Escalabilidade:** Estrutura preparada para crescimento futuro
- **Manutenibilidade:** Código limpo, testado e bem documentado
- **Zero débito técnico:** Ferramentas modernas evitam legacy code

13.2 Próximos Passos

1. Criar repositório GitHub com template do monorepo
2. Configurar pnpm workspaces e Turborepo
3. Implementar @code-guardian/core seguindo a estrutura definida
4. Desenvolver extensão VS Code com foco em UX não-intrusiva
5. Validar MVP com projeto real antes de adicionar features extras

Lembre-se: O MVP é deliberadamente limitado para permitir validação rápida. Resistir à tentação de adicionar features prematuras é essencial para o sucesso do projeto.