Conceitos Fundamentais Engenharia Estrutural VS CODE

1. Conceitos Fundamentais de Workspace & Pastas

- Um *workspace* no VS Code é basicamente uma pasta aberta ou um arquivo de workspace (.code-workspace) que agrupa uma ou mais pastas de projeto. Visual Studio Code+1
- No modo padrão, abre-se uma pasta única como workspace. Mas existe o modo *multi-root workspace*, permitindo várias pastas num único workspace. Visual Studio Code+1
- A pasta raiz ou as pastas configuradas em .code-workspace servem como "contexto" para configurações específicas, tarefas, depuração e extensões. Visual Studio Code

---

2. Organização Recomendada de Pastas & Arquivos para o Projeto RAWN PRO

Para seu projeto (front-end + back-end + infra + docs) recomendo a seguinte estrutura de pastas:

/rawnpro-project/

```
├─ /api/            ← backend (.NET / Node)
│   ├─ src/
│   ├─ tests/
│   └─ Dockerfile (se aplicável)
├─ /web/            ← front-end (PWA React/Next)
│   ├─ src/
│   ├─ public/
│   └─ next.config.js (ou equivalente)
├─ /infra/          ← infra/CI/CD scripts, configs (GitHub Actions, Vercel)
├─ /docs/           ← documentação (specs, arquitetura)
├─ /.vscode/        ← configuração padrão de workspace (explained abaixo)
│   ├─ settings.json
│   ├─ extensions.json
```

│ ├─ tasks.json

│ └─ launch.json

├─ rawnpro.code-workspace ← arquivo de workspace (multi-root se necessário)

├─ package.json (ou equivalente root)

└─ README.md

Justificativas:

- Separação clara entre front-end, back-end e infra.
- Pasta .vscode para **configurações compartilhadas** do editor (tasks, debugger, extensões).
- Arquivo rawnpro.code-workspace permite definir multi-root caso queira que editor reúna /api, /web, /infra como pastas distintas de um só workspace. Visual Studio Code
- Documentação em /docs/ para manter specs, design system e decisões (como stack, prompt, arquitetura).
- Infra scripts isolados para CI/CD e deployment.

---

3. Arquivos de Configuração Essenciais no VS Code

.vscode/settings.json

- Contém definições específicas do workspace (indentação, formatação, exclusões de arquivos). Visual Studio Code
- Exemplo:
- {
- "editor.tabSize": 2,
- "files.exclude": {
- "**/node_modules": true,
- "**/.git": true
- },
- "eslint.enable": true
- }

extensions.json

- Lista de extensões recomendadas ou obrigatórias para o projeto, para garantir consistência de ambiente na equipe. Visual Studio Code

tasks.json

- Define tarefas automatizadas (build, test, lint) integradas ao VS Code.

launch.json

- Configurações de depuração para front-end/back-end (breakpoints, attach, environment variables).

rawnpro.code-workspace (no caso multi-root)

- Estrutura JSON com entradas de pastas:
- {
- "folders": [
- { "path": "api" },
- { "path": "web" },
- { "path": "infra" }
-],
- "settings": {
- "files.exclude": {
- "**/node_modules": true
- }
- },
- "extensions": {
- "recommendations": ["dbaeumer.vscode-eslint", "esbenp.prettier-vscode"]
- }
- }

Isso facilita compartilhar ambiente completo de projeto. Visual Studio Code

---

4. Camadas e Responsabilidades de Pastas

- **/api/**: Back-end. Contém código de lógica servidor, endpoints, orquestração com modelo IA, integração com Kiwify, autenticação, dados (perfil do usuário).
- **/web/**: Front-end. UI PWA, componentes de chat, integração com API back-end, experiência mobile-first.
- **/infra/**: Scripts de infraestrutura (deploy Vercel, integração GitHub, docker-compose, configuração de ambientes dev/prod).
- **/docs/**: Documentação técnica, arquitetura, design system, fluxos de conversa, requisitos.
- **/.vscode/**: Configurações do editor para padronização da equipe.
- **Root arquivos**: README com visão global do projeto, arquivo .gitignore, scripts de inicialização.

---

5. Boas Práticas de Organização

- Use nomenclatura clara: api, web, infra para que novos membros entendam.
- Cada camada deve ter README específico explicando sua função.
- Evite arquivos de configuração misturados com código funcional; mantenha config em .vscode ou infra.
- Versione .vscode e *.code-workspace para garantir consistência de ambiente.
- Utilize files.exclude e search.exclude nas configurações para ocultar pastas pesadas/excluídas (ex: node_modules, build). DeepWiki
- Se o projeto crescer em micro-serviços ou módulos, considere sub-pastas como /services/, /shared/ etc.

---

6. Integração com Controles de Versão e Deploy

- O repositório deve conter todo o código das camadas mencionadas.
- .vscode e .code-workspace podem estar no repositório para padronização.
- A pasta /infra/ deve conter definições de CI/CD e deploy, que podem ser acionadas via tasks do VS Code.

- O front-end /web/ integrado com Vercel para deploy automático. Back-end /api/ integrado com GitHub Actions ou outro pipeline.

---

7. Checklist Técnico para o Engenheiro

- Criada estrutura de pastas conforme o modelo acima.
- Adicionado arquivo rawnpro.code-workspace com pastas definidas.
- Configurado .vscode/settings.json, extensions.json, tasks.json, launch.json.
- Configurado files.exclude para ocultar pastas irrelevantes.
- Estabelecido padrões de nomes, readmes e documentação para cada camada.
- Integrado tasks de build/test no VS Code.
- Verificado que ao abrir VS Code com code . a estrutura de workspace carrega corretamente.
- Instruído equipe para usar este workspace padrão e sincronizar configurações.
- Garantido que ambiente remoto ou dev container (se aplicável) respeite esta estrutura.