# Diretrizes de Engenharia e DevOps

**Projeto:** Gestão de Saúde Ocupacional (SaaS) **Responsável:** Gaj

Este documento estabelece as regras de ouro para o desenvolvimento, versionamento e deploy do sistema. O objetivo é manter a base de código limpa, segura e escalável, evitando débito técnico.

## 1. Fluxo de Trabalho (Git Workflow)

Adotamos o **GitHub Flow Simplificado**. A branch `main` é sagrada e deve conter sempre código testado e pronto para produção.

### 1.1. Regras de Branch

- **Nunca** faça commit diretamente na `main`.
- Crie uma branch para cada nova funcionalidade ou correção.

### 1.2. Padrão de Nomenclatura

Use prefixos para categorizar o trabalho:

- `feat/nome-da-funcionalidade` (Ex: `feat/cadastro-funcionario`) -> Novas features.
- `fix/nome-do-erro` (Ex: `fix/calculo-validade-aso`) -> Correções de bugs.
- `docs/nome-do-doc` (Ex: `docs/readme-update`) -> Apenas documentação.
- `refactor/nome-da-melhoria` -> Melhoria de código sem mudar funcionalidade.

### 1.3. Ciclo de Vida da Tarefa

1. Atualize sua base: `git checkout main` e `git pull`.
2. Crie a branch: `git checkout -b feat/nova-tela`.
3. Desenvolva e faça commits locais.
4. **Antes de fechar:** Rode o build local (`npm run build`) para garantir que não quebrou nada.
5. Faça o Merge para a `main`.

## 2. Padrão de Commits (Conventional Commits)

Escreva mensagens de commit que contem uma história. Isso gera um histórico limpo.

**Estrutura:** `tipo(escopo): descrição curta`

- `feat(operacional): adiciona tela de novo exame`
- `fix(financeiro): corrige soma total da fatura`
- `style(ui): ajusta cor do botão primário`

- `chore(deps): atualiza versão do nextjs`

**O que evitar:**

- ❌ "ajustes"
- ❌ "up"
- ❌ "corrigindo bug"

## 3. Estratégia de Testes (QA)

Como a equipe é enxuta, não buscamos 100% de cobertura. Focamos em **Testes Cirúrgicos** nas lógicas críticas.

### 3.1. O que testar OBRIGATORIAMENTE (Testes Unitários/Vitest)

Se a lógica envolver **Dinheiro**, **Datas** ou **Segurança**, deve ter teste automatizado.

1. **Cálculo de Validade do ASO:** Garantir que Risco 3 + Idade X gere a data correta.
2. **Snapshot de Preços:** Garantir que o sistema está gravando o valor histórico, não o atual.
3. **Somatória de Faturas:** Garantir que a soma dos itens bate com o total.

### 3.2. O que testar MANUALMENTE

- Fluxos de tela (UX).
- Responsividade (Mobile/Desktop).
- Cadastro simples (CRUDs).

## 4. Banco de Dados (Supabase)

### 4.1. Migrations e Alterações

- Nunca altere o banco de produção (Supabase Web UI) manualmente para mudanças estruturais.
- Crie as tabelas localmente ou via SQL Editor e salve o script na pasta `/supabase/migrations` ou `/docs/database`.
- Mantenha o arquivo `schema.sql` atualizado no repositório.

### 4.2. Segurança

- **JAMAIS** suba o arquivo `.env.local` para o GitHub.
- Chaves `SERVICE_ROLE` (admin total) nunca devem ser usadas no Frontend.

## 5. Deploy e CI/CD (Vercel)

O deploy é contínuo e automático.

- **Gatilho:** Qualquer push na branch `main` dispara o deploy na Vercel.
- **Regra de Ouro:** "Se está na main, está no ar."

- Não suba código incompleto para a main. Se precisar salvar trabalho incompleto, mantenha na branch `feat/` .

#### 5.1. Verificação de Pré-Deploy

Antes de dar o merge na main, verifique:

1. O projeto roda o build sem erros? ( `npm run build` )
2. O Lint não está gritando erros? ( `npm run lint` )
3. As variáveis de ambiente novas foram adicionadas na Vercel?

## 6. Definição de Pronto (DoD - Definition of Done)

Uma tarefa só é considerada concluída quando:

- [ ] O código foi escrito.
- [ ] O código foi formatado (Prettier/ESLint).
- [ ] Testes unitários (se aplicável) passaram.
- [ ] A funcionalidade foi testada manualmente no navegador.
- [ ] O código foi mergeado na `main` e o deploy na Vercel ficou verde (Sucesso).