

24-08-25 : 20:01

Tags:

Status:

Autor: Vitor Hugo

Reunião Guia

1. FOCO

- Tarefas não documentadas
 - Sem quebra dos problemas
 - Sem Definição de Pronto
 - Dispersão do foco dos problemas atuais
 - Citar técnica de Valor x Esforço
- Projeto sem escopo claro
 - Muitas mudanças do objetivo final em pouco tempo
 - Atrapalha devido a grande quantidade de tarefas
 - Dispersão do foco dos problemas atuais
- Desgaste com soluções futuras
 - Planejamento de estruturas complexas no qual o uso ainda não existe ou é pouco aproveitável
 - Dificulta o foco em soluções simples e práticas

2. Arquitetura

2.1 Arquitetura Macro (Sistema)

Funcionalidade, componentes, relações e fluxo de dados.

- **Componentes:** Serviços, APIs, Front-end, Back-end, bancos de dados, filas e caches.

- **Comunicação:** Como os componentes se comunicam (REST, eventos, GraphQL.)
- **Padrões:** Microsserviços, DDD, etc.
- **Fluxo de Dados:** Como a Informação passa por todos os componentes do sistema.

Cada nova aplicação sem escopo definido impacta diretamente em todo o sistema e sua comunicação tornando a arquitetura inflexível e o risco é que os serviços fiquem frágeis a cada nova alteração.

- **Exemplo:** Botão de edição de mensagem do Whatsapp + clusters de banco de dados

2.2 Arquitetura da Infraestrutura

Provisionamento, escalabilidade, disponibilidade, confiabilidade e segurança da infraestrutura.

CPX31 = 4vcpu + 8gb RAM

CPX21 = 3vcpu + 4gb RAM

- **Arquitetura Proposta (3 servidores):**
 - **Core (CPX31 ou CPX21) \$18.59 ou \$10.59:**
 - **Coolify (Redis + Postgres)**
 - Facilidade de deploy com git e github
 - SSL (Let's Encrypt) + Traefik automáticos
 - Diminuição da complexidade de infra
 - Multi-server
 - **Monitoramento:**
 - **Uptime Kuma** → Monitoramento de disponibilidade
 - Monitoramento com Sentinel e Metrics
 - **Logs e Backups:**
 - Backups Nativos com S3 para Coolify e Postgres
 - Logs nativos por servidor e recurso Drain Logs
 - **Aplicações Internas (CPX31) \$18.59:**
 - **N8N Queue Mode:**

- Instâncias para Editor, Webhook e Worker
- Redis + Postgres
- Via Docker-compose
- **Typebot + MinIO:**
 - Typebot Builder e Viewer
 - MinIO Storage
- **ZapMaster:**
 - Via docker ou clone github
- **CRM masterClassic interno (Opcional):**
 - Sistema de Produtos
 - Documentação ou Gitbook.
- **Aplicações Públicas (CPX31) \$18.59:**
 - Site
 - Formulários
 - CRM modo SaaS
 - Portal de Parceiros
- **Supabase (Plano PRO) \$25:**
 - Backups diários guardados por 7 dias.
 - Retenção de logs por uma semana
 - 8GB de disco
 - Usar o plano gratuito para testes e novas aplicações
 - Diminui complexidade de infra não necessita manutenção.
 - Considerar self hosting só quando o custo do supabase aumentar

Pontos de Atenção | Riscos | POF:

- Indisponibilidade do Supabase
- Sobrecarga do Servidor 2 caso fluxos do n8n aumentarem (50+)
- Rede Hetzner falhar
- Expiração do certificado SSL

Custo Total: \$ 72,77 - \$ 80,77 | R\$ 392,95 - R\$ 436,15

Por que ?

- Coolify reduz trabalho em DevOps, SSL automático, instalação simplificada, deploy automático.
- Supabase elimina manutenção e necessidade de infraestrutura adicional.
- Separação em 3 servidores mantém os serviços disponíveis e reduz POF sem aumentar muito os custos com gerenciamento centralizado.
- Foco no desenvolvimento
- Portainer → ótimo para gerenciamento avançado de containers docker, uso de kubernetes e clusters, mas exige conhecimento avançado para aproveitar de todas as features.
- Flexibilidade para migração ou escalar.

Ambiente de Dev x Prod

- **Separação Lógica:**
 - Separação através de environments no Coolify:
 - Mais simples e barato
 - Consome recursos do servidor
- **Separação Física:**
 - Inclusão de mais um servidor para Ambiente Dev.
 - Aumento de complexidade e configurações
 - Forte isolamento de ambientes
 - Não consome recursos de aplicações usadas

2.3 Arquitetura do Código

Estrutura de pastas, módulos, classes, design patterns, princípios de código limpo e testabilidade.

Como a funcionalidade é implementada?

- Estrutura de Projeto: MVC, Clean, camadas
- Design Patterns
- SOLID: Princípios de design de software
- Convenções: Nomenclaturas e estilos de código

- Gerenciamento de dependências
- Testes

3 - Desenvolvimento

IA

- Funciona com os códigos abertos na internet
- Segurança do código (XSS, SQL injection, exposição de chaves, dependências duvidosas)
- Persua → 90% do código feito por IA mas com bugs de segurança
- Não tem distinção nem consideração acerca de ferramentas (bun.js x NodeJs)
- Falta de padrão e contexto de negócio.

Stack Alvo - Next.js + React + TypeScript

- Versões, estabilidade e consistência.
- Baixa curva de aprendizado → equipe conhece JS
- Backend in Frontend → API routes
- Rotas automáticas, SSR SSG, CSR e ISR.
- Otimização de recursos web, SEO otimizado.
- Middlewares e deploy simplificado.

Python x PHP x Next.Js

- Citar problemas de segurança do PHP.
- Python para IA, automações, scripts
- NextJS foi criado para web rápida, escalável e flexível.
- Usado em aplicações de dados em tempo real.

3.1 Git

- Separação de Branches: main, dev
- Conventional Commits

- SemVer → vX.Y.Z
- Deploy via Coolify

3.2 Padrões

- Estrutura de Pastas → Arquitetura por features
- SRP

3.3 Ambiente de Dev

- Git
- VsCode → Ext: ESLint, Prettier, GitLens (ou outra), Docker
- Cursor/Copilot
- Windows: Chocolatey + execução de scripts habilitada
- Docker Desktop
- Python: pyenv-win + pipenv
- Node.js: nvm
- Github Desktop Opcional
- Configuração dos repositórios remotos

4 - Propostas e plano de ação

- Resolver o cadastro no LIS da forma mais prática e simples.
- Resolver o backend/fluxo do pix automático.
- Finalização da migração
- Banco de Dados
- Estruturar os fluxos de trabalho e metodologia
- Refatorar as aplicações e realizar melhorias
- Começar a desenvolver automações e aplicações internas.