**ADR 001 - Estrutura e Tecnologias da Aplicação URL\_SHORTENER\_API**

**Status:** Decidido  
 **Data:** 2025-06-06  
 **Decisores:** Carlos Henrique Parreira Jacinto

### **Contexto:**

Estou desenvolvendo uma API para encurtamento de URLs que precisa ser modular, escalável e com suporte a testes automatizados e deploy via contêineres.

### **Decisão:**

Optamos por utilizar as seguintes tecnologias e práticas:

**1. NestJS com TypeScript:**  
 Framework principal da API.  
 Estrutura modular com divisão clara entre domínios (modules/auth, modules/users, modules/urls).  
 Foco em boas práticas como Inversão de Dependência, DDD e separação de responsabilidades (DTO, Services, Controllers, etc).

**2. Prisma ORM**  
 Abstração da camada de acesso a dados usando Prisma.  
 Integração direta com banco PostgreSQL.

**3. PostgreSQL via Docker**  
 Banco de dados relacional utilizado.  
 Substitui necessidade de instalação local.  
 Configurado no docker-compose-dev.yml.

**4. Docker**  
 Containerização total da aplicação (API, DB).  
 Arquivo Dockerfile define imagem da aplicação.  
 entrypoint.sh para facilitar inicialização do container.  
 Ambiente de desenvolvimento controlado e replicável.

**5. Jest**  
 Framework de testes automatizados.  
 Estrutura de testes por módulo (/test dentro de cada domínio).  
 Configurado em jest.config.ts.

**6. Arquitetura Modular**  
 Organização em /src/modules por domínio (users, urls, auth).  
 Pastas internas para dto, services, test e utils quando necessário.  
 Uso do padrão de injeção de dependência do Nest.

**7. Observabilidade com Prometheus e Grafana**  
 Coleta de métricas da aplicação com Prometheus.  
 Visualização e análise de métricas em tempo real com Grafana.  
 Facilita monitoramento da saúde da aplicação e identificação de gargalos de performance.

**8. Documentação com Swagger (OpenAPI)**  
 Geração automática da documentação da API com Swagger.  
 Disponível via endpoint /api para facilitar testes e integração com terceiros.  
 Mantida em conformidade com os DTOs e decorators do NestJS.

### **Consequências:**

Facilita manutenção e crescimento da aplicação.  
 Permite integração contínua com pipelines Docker.  
 Modularidade promove testabilidade e legibilidade.  
 Permite monitoramento contínuo da aplicação em produção.  
 Facilita integração de times externos via documentação interativa.  
 Requer atenção ao versionamento de dependências e sincronização dos containers.