

Padrão de Arquitetura de Sistemas

Sumário

1	Introdução.....	3
2	Tecnologias	3
2.1	Componentes Fundamentais da Arquitetura.....	3
3	Padrões Arquiteturais para Desenvolvimento de Soluções.....	4
3.1	Linguagens de programação	4
	Banckend	5
	Frontend.....	5
3.2	Itens Arquiteturais Padrões.....	6
	Aplicação Básica.....	6
	Armazenamento de Dados.....	7
	Comunicação e Integrações	8
	Segurança	8
	Logs de Sistema e Tratamento de Erro	8
	Auditoria de Dados de Negócio	9
	Armazenamento de Arquivos	10
3.3	Padrões de Desenvolvimento GOV.BR	10
	Frontend.....	11
	Privacidade e Segurança da Informação (PPSI).....	11
	Interoperabilidade - ePING.....	11
	Integração ao Login Único (GOV.BR).....	12
4	Padrões de Projeto para Desenvolvimento de Sistemas.....	12
4.1	Padrões de Projeto	12
5	CI/CD – Integração Contínua e Entrega Contínua	17
5.1	Visão Macro do Fluxo CI/CD.....	17
5.2	Conceitos Principais do CI/CD	20
	Integração Contínua (CI – Continuous Integration)	20

Implantação Contínua (CD – Continuous Deployment)	21
5.3 Componentes Essenciais de um Pipeline CI/CD.....	21
Versionamento de Código (SCM - Source Control Management)	21
Automação do Build	21
Testes Automatizados	22
Construção e Distribuição de Imagens Docker	22
Implantação com Kubernetes e Rancher	22
6 Dados	23
6.1 Considerações de Modelagem Específicas	23
6.2 Relacionamentos e Integridade Referencial	25
6.3 Técnicas de Otimização para Altos Volumes	25
6.4 Volumetria	25
6.5 Internalização e Hospedagem	27
7 Referências	28

1 Introdução

Este documento define o padrão arquitetural para o desenvolvimento, manutenção e internalização de sistemas da Agência Nacional de Transportes Terrestres - ANTT. O objetivo é garantir a consistência, escalabilidade, segurança e manutenibilidade dos sistemas, alinhando-se às melhores práticas e padrões do mercado.

As definições aqui apresentadas são de uso obrigatório. Caso exista opção mais adequada para determinada demanda, a Superintendência de Tecnologia da Informação - SUTEC deve ser consultada para avaliar a possibilidade de sua adoção.

2 Tecnologias

2.1 Componentes Fundamentais da Arquitetura

A tabela 1 descreve o propósito e as responsabilidades de cada componente, ou seja, ferramentas, que são utilizadas por padrão na ANTT.

COMPONENTE	DESCRIÇÃO E OBJETIVO	FERRAMENTAS	OBSERVAÇÕES
Camada de Serviços/Backend	Contém os microservices e backend-API responsáveis pela lógica do negócio.	.NET 10 ou .NET 8	Isolamento funcional com comunicação REST/JSON e integração com databases.
Servidor de Autenticação	Implementa autenticação e autorização centralizada usando padrões OIDC e OAuth 2.0.	Keycloak	Integração direta com o Keycloak e emissão de tokens JWT seguros.
Camada Distribuída	Implementa replicação e alta disponibilidade com relays e caching distribuído.	Redis ou Memcached	Reduz as latências e melhora o desempenho de sistemas distribuídos.
Mensageria Distribuída	Gerencia mensagens assíncronas e filas para desacoplar dependências entre serviços.	RabbitMQ ou Kafka	Utilizado principalmente para eventos interserviços e jobs agendados.
Visualização de Métricas	Gera gráficos e dashboards interativos com dados de métricas coletadas.	Grafana	Integrado ao Zabbix para visualizações avançadas das métricas do sistema.

COMPONENTE	DESCRIÇÃO E OBJETIVO	FERRAMENTAS	OBSERVAÇÕES
Centralização de Logs	Agrega todos os logs de microservices, backend API e outros componentes.	Elasticsearch, Logstash e Kibana	Usados conjuntamente para indexação (Elasticsearch), streaming (Logstash) e visualização (Kibana).
Base de dados relacionais	Armazenamento de dados relacionais	PostgreSQL (principal) e SQL Server	Armazenamento dos dados relacionais
Base de dados não relacionais	Armazenamento de dados NoSql	MongoDB	Armazenamento de dados não relacionais (NoSQL) orientado a documentos
Camada de apresentação/frontend	Contém frontend das aplicações se comunicando com o(s) microservices e/ou backend-API, via REST-API (json)	ReactJs, Primereact, React Native	Camada de apresentação das aplicações Web e Mobile. Deve seguir design system recomendado pelo GOV.BR

Serviços de monitoramento utilizadas dentro da ANTT

SERVIÇO	DESCRIÇÃO
Dynatrace	Plataforma de monitoramento de performance de aplicações (APM) que captura e unifica as dependências entre todos os dados de observabilidade para combinar métricas, logs, traços e dados de experiência do usuário com inteligência.
Zabbix	O Zabbix é um software que monitora vários parâmetros da rede, dos servidores e da saúde dos serviços. Utiliza-se de um mecanismo flexível de notificação que permite configurar alertas por e-mail para praticamente qualquer evento. As notificações permitem que se reaja rapidamente a problemas no ambiente

3 Padrões Arquiteturais para Desenvolvimento de Soluções

3.1 Linguagens de programação

A linguagem de programação padrão para o desenvolvimento backend é o C# na plataforma .NET 8. Para o frontend deve-se utilizar a linguagem TypeScript na plataforma NodeJs.

Banckend

Backend é a parte de uma aplicação que não é visível ao utilizador, mas que é responsável por fazer a aplicação funcionar. É também conhecido como lado do servidor. Por padrão, é adotada na ANTT a linguagem C# que deverá estar de acordo com a versão da plataforma .NET recomendada pela [Microsoft](#) (vide referências).

Frameworks mais utilizados:

- ORM: EntityFramework, onde o mapeamento de toda as entidades relacionamentos devem ser devidamente definidas, onde deve ser explicitamente mapeados nome de entidade, nome de campo, tipo do campo, tamanho do campo, sua obrigatoriedade;
- Serilog: para gerenciamento de logs de sistemas.
- Elastic.Serilog.Sinks: Para logs a serem enviados para Elasticsearch;
- Swashbuckle.AspNetCore: utilizado para criação e documentação dos endpoints de uma REST-API (Swagger);
- AutoMapper: usado para parse entre objetos;
- FluentValidation: para criar validações de objetos, de forma a facilitar a criação de validações e seus retornos;
- Refit: utilizado para automatizar, de forma type-safe, o mapeamento dos endpoints de uma REST-API para interfaces C#;
- JWT: utilizado como formato padrão para token de autenticação e autorização;

Frontend

Frontend é a parte visual de uma aplicação ou site, onde o utilizador interage com os elementos visuais. O padrão de frontend utilizado na ANTT, é React, onde a linguagem utilizada é o Typescript.

Abaixo estão os Frameworks mais utilizados na Agência:

- Primereact: componentes prontos, e práticos para customização e criação de novos;
- Redux: um centralizador e gerenciador do estado e lógica da aplicação;

- Axios: biblioteca para comunicação REST-API;
- StyledComponents: componente para trabalhar a estilização (css) da aplicação, de forma prática para trabalhar com React;

3.1.1.1 Padrões de UI/UX

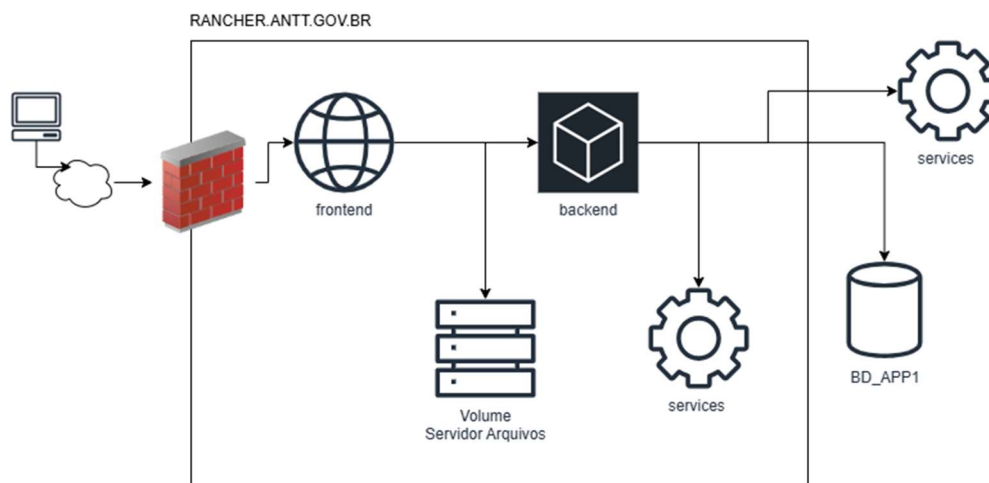
A camada de apresentação das aplicações Web e Mobile da ANTT deve seguir os padrões definidos pelo [Padrão Digital de Governo](#), assim como implementar padrões de acessibilidade, conforme as diretrizes do [eMAG](#).

3.2 Itens Arquiteturais Padrões

Na definição da arquitetura de uma solução específica deve-se utilizar itens arquiteturais já consolidados na ANTT, tais como: armazenamento de dados, arquivos, desenvolvimento e implantação em containers, segurança, logs, comunicação e serviços.

Aplicação Básica

Em uma aplicação básica, independentemente de sua complexidade, o frontend é separado fisicamente do backend. Ambos devem ser criados em containers, com sistema operacional Linux, para serem implantados nos ambientes de desenvolvimento, testes, homologação e produção.



Além da comunicação entre frontend e backend, seguem abaixo outras comunicações/definições padrões:

- Apenas o frontend é exposto externamente;
- O armazenamento de arquivos deve ser feito através de mapeamento de volume no container;
- Pode haver serviços dentro e fora do ambiente rancher;
- As bases de dados são criadas em servidores fora ambiente do rancher.

Armazenamento de Dados

O banco de dados padrão a ser utilizado é o PostgreSQL (versão mínima 15) ou o MS SQLServer (versão mínima 2019).

O acesso/persistência deve ser feito através do ORM Entity Framework, implementado na camada de Infrastructure/Data.

As definições mais técnicas referentes a cada SGBD devem ser alinhadas junto à equipe de banco de dados da área de infraestrutura de TIC da ANTT.

Em relação às normas, nomenclaturas e padrões de criação do modelo do banco de dados, deve ser utilizado o documento [Normas e Padrões de Administração de Banco de Dados da ANTT v1.7-2.docx](#) ou versão mais atual.

Nos casos de sistemas desenvolvidos externamente, a escolha do SGBD deve ser feita após alinhamento prévio e validação da SUTEC.

Comunicação e Integrações

Para atender aos requisitos funcionais e não-funcionais da aplicação, a arquitetura base se integra aos sistemas de controle de acessos, envio de e-mails, consumo de dados de CPF, CNPJ, Município, UF, login gov.br, CorporativoServices, dentre outros.

Segurança

O sistema deve utilizar o padrão de autenticação e autorização via token (JWT, Microsoft.AspNetCore.Authentication), seguindo o padrão OIDC (OpenID Connect/OAuth2) para implementar os requisitos de segurança do sistema, como autenticação e autorização, com base nas informações obtidas através da integração com o sistema KeyCloak.

Na arquitetura de solução, por sua vez, deve-se utilizar RBAC - controle de acesso baseado em perfis - para gerenciar os níveis de acessos às funcionalidades.

A comunicação da aplicação e serviços de autenticação/autorização deve ter as credenciais trafegadas de forma criptografada, ou seja, o frontend deve criptografar as credenciais antes de encaminhá-las para o backend, e o backend deve também encaminhar as credenciais criptografadas para o serviço de autenticação.

No que diz respeito à disponibilização de serviços (REST-API) para consumo por sistemas externos à ANTT, apenas os endpoints e dados necessários ao consumo devem ser expostos. Este serviço (REST-API) deve ser construído como uma fachada, que, por sua vez, consumirá a API backend e fornecerá os dados ao cliente.

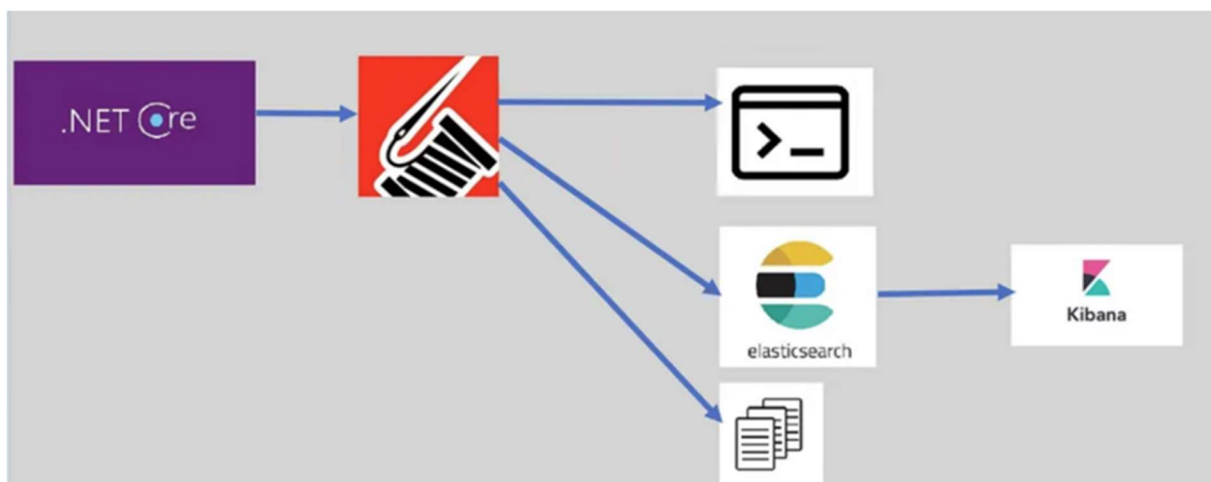
Logs de Sistema e Tratamento de Erro

Logs de sistema são arquivos de texto que registram eventos de um sistema operacional, aplicação ou serviço, como erros, alertas, desligamentos inesperados e mensagens de inicialização.

Nesse sentido, o backend desenvolvido em .NET deverá utilizar a biblioteca Serilog para registros de logs da aplicação. Nela é possível definir o nível de logs que será registrado podendo ser de Information, Warning ou Error.

Os logs devem ser escritos no console do container da API, na pasta logs do storage e enviado para o servidor do ElasticSearch.

O ElasticSearch, por sua vez, recebe as informações dos logs através do Elastic.Serilog.Sinks e disponibiliza no dashboard do Kibana.



O backend tem como responsabilidade fazer o gerenciamento de vida dos arquivos de logs, respeitando as seguintes regras básicas:

- O arquivo deve ser gerado e nomeado pelo nome da aplicação, data e hora;
- Em caso de necessidade de criar mais de um arquivo na mesma hora, deve ser acrescentado um sequencial;
- O tamanho do arquivo não deve passar de 10Mb;
- O tempo de retenção dos arquivos deve ser entre 15 e 30 dias;
- Devem ser criadas variáveis de ambientes para mudança do nível de logs a serem gerados e enviados (Information, Warning, Error).

Auditoria de Dados de Negócio

A arquitetura dos sistemas deve incluir, como parte de sua solução, a auditoria de dados. Todas as operações de persistência (inclusão, alteração, exclusão lógica ou física) realizadas em tabelas designadas para auditoria devem ser registradas em

um log de auditoria de dados. A seguir, estão os dados que devem ser registrados nesses logs:

- Quem realizou a operação, ou seja, o usuário que está logado no sistema;
- A operação realizada persistência (incluir, alterar, exclusão);
- Quando foi realizada a operação (data/hora);
- Os dados que sofreram a ação, IP (externo) do usuário logado que realizou a operação;
- Sistema que realizou a operação;
- Usuário de banco (interno) que realizou a operação;
- IP (interno) do usuário que realizou a operação.

O registro de dados de auditoria deve ser armazenado em uma transação atômica garantindo a persistência dos dados conforme a operação realizada com sucesso.

Armazenamento de Arquivos

Quando a solução incluir o armazenamento de arquivos, deverão ser adotadas as seguintes regras:

- Os arquivos não devem ser armazenamento todos em uma única pasta;
- A quantidade de arquivos não deve exceder a quantidade de 10.000 em uma única pasta;
- O nome do arquivo não pode conter acentuação, caracteres especiais e espaço em branco, com a exceção dos caracteres '-' e '_';
- O volume de armazenamento deve ser fornecido pela área de infraestrutura da SUTEC.

3.3 Padrões de Desenvolvimento GOV.BR

Os projetos e aplicações da ANTT devem seguir os padrões de apresentação, segurança, políticas de privacidade de dados e integrações definidos pelo Sistema de Administração dos Recursos de Tecnologia da Informação (Sisp).

Frontend

A camada de apresentação das aplicações Web e Mobile da ANTT devem seguir os padrões recomendados pelo Padrão Digital de Governo, bem como implementar padrões de acessibilidade conforme as diretrizes do eMAG.

Privacidade e Segurança da Informação (PPSI)

A Privacidade e Segurança da Informação no Governo Digital tem como ponto de partida o Programa de Privacidade e Segurança da Informação (PPSI), o qual envolve um conjunto de ações de adequação na temática.

Dessa forma, é importante possuir e manter trilhas de qualidade e teste de software, e realizar desenvolvimento seguro, aderente ao que for disposto em dispositivos legais correlatos em publicações feitas pelo GSI/PR.

Além disso, as aplicações devem seguir as boas práticas estabelecidas pela Lei Geral de Proteção de Dados (LGPD), tanto na coleta de dados por meio de aplicações Web e Mobile, quanto nas integrações com sistemas externos à Agência e no armazenamento dos dados.

Ademais, considerando a evolução e modernização dos ataques cibernéticos, é imprescindível que esta Agência disponibilize soluções/sistemas capazes de lidar com eventuais ataques, por meio de técnicas de desenvolvimento seguro, além de seguir as boas práticas de mercado, como a OWASP top 10. Essas medidas contribuem para reduzir vulnerabilidades em sistemas e aplicações, assegurando que desde a concepção de novos projetos a segurança seja tratada como prioridade.

Interoperabilidade - ePING

A criação de serviços para integração deve seguir, além das boas práticas de segurança, o padrão interoperabilidade de governo eletrônico (ePING). A integração com sistemas governamentais deve ter a capacidade de interoperar diversos sistemas e organizações que trabalham em conjunto, de modo a garantir que pessoas, organizações e sistemas computacionais interajam para trocar informações de maneira eficaz e eficiente.

A arquitetura [ePING](#) – Padrões de Interoperabilidade de Governo Eletrônico – define um conjunto mínimo de premissas, políticas e especificações técnicas que regulamentam a utilização da Tecnologia de Informação e Comunicação.

Integração ao Login Único (GOV.BR)

As aplicações que são acessadas por usuários externos à Agência, seja um cidadão, instituição privada ou governamental, devem implementar Single Sign-On (SSO) utilizando o sistema de login único do Governo Federal via Keycloak.

4 Padrões de Projeto para Desenvolvimento de Sistemas

Esta seção define os padrões de projetos utilizados no desenvolvimento das aplicações, suas organizações e necessidades a serem implementadas.

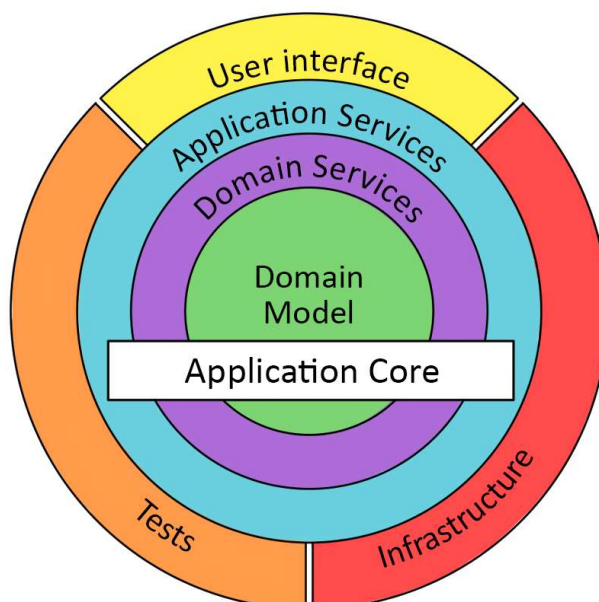
4.1 Padrões de Projeto

Padrões de projeto (design patterns) são soluções típicas para problemas comuns em projeto de software. Cada padrão é como uma planta de construção que você pode customizar para resolver um problema de projeto particular em seu código.

Para a implementação do backend em .NET, é recomendado seguir o padrão de projeto DDD (Domain Driven Design – Eric Evans), onde o negócio e suas regras são centralizados no domínio. Seguindo este padrão, abaixo é apresentada a nomenclatura e estrutura do projeto:

- Api – controllers que definem a interface com usuário;
- Application – segregação de responsabilidade entre leitura e escrita, orquestração do fluxo de dados para as camadas de serviço;

- Domain – entidades de domínio, interfaces, serviços e regras de negócio;
- Infrastructure
- CrossCutting
- ServiceAgents – integrações com sistemas internos e externos;
- Validation – validações;
- Data – acesso e persistência aos dados;
- SharedKernel – interface padrão para implementação dos services, repositories, unitOfWork, dataContext e responseQueryResult;



O acrônimo SOLID representa os cinco princípios que facilitam o processo de desenvolvimento — o que facilita a manutenção e a expansão do software. Estes princípios são fundamentais na programação orientada a objetos e podem ser aplicados em qualquer linguagem que adote este paradigma.

Os 5 princípios são:

S — Single Responsibility Principle (Princípio da responsabilidade única)

O — Open-Closed Principle (Princípio Aberto-Fechado)

L — Liskov Substitution Principle (Princípio da substituição de Liskov)

I — Interface Segregation Principle (Princípio da Segregação da Interface)

D — Dependency Inversion Principle (Princípio da inversão da dependência)

Todas as classes e namespaces devem ser pensados e construídos dentro de um padrão de responsabilidade, seguindo os princípios SOLID. Uma arquitetura que combina os padrões DDD e os princípios SOLID organiza o sistema em camadas bem definidas, promovendo coesão, escalabilidade e facilidade de manutenção.

No DDD, a aplicação é estruturada em domínio, aplicação, infraestrutura e interface, com o foco central nas regras de negócio representadas no domínio. Os princípios SOLID asseguram que cada classe tenha uma única responsabilidade (SRP), facilite extensões sem modificar comportamentos existentes (OCP), dependa de abstrações em vez de implementações (DIP), e promova o uso adequado de interfaces e herança.

Essa combinação resulta em um sistema com baixo acoplamento, alta coesão e alinhado aos conceitos do negócio, permitindo escalabilidade, testes isolados e uma base sólida para evolução contínua.

Padrões de DDD com aplicação de SOLID, com exemplo em .NET:

- Entidades

DDD: Representam objetos com identidade única e regras de negócio específicas.

SOLID - SRP (Single Responsibility Principle): Cada entidade foca em representar um único conceito do domínio, mantendo suas responsabilidades coesas.

Exemplo no .NET:

```
public class Cliente
{
    public Guid Id { get; private set; }
    public string Nome { get; private set; }
    public Cliente(string nome) => Nome = nome;
}
```

- Objetos de Valor (Value Objects)

DDD: Representam conceitos imutáveis do domínio, definidos apenas pelos seus atributos.

SOLID - SRP: A lógica de validação ou comportamento específico fica encapsulada no objeto de valor, separando responsabilidades.

SOLID - LSP (Liskov Substitution Principle): Objetos de valor podem ser substituídos por outros sem alterar o comportamento geral.

Exemplo no .NET:

```
public record CPF(string Numero)
{
    public CPF Validate()
    {
        // Validação específica do CPF
        return this;
    }
}
```

- Agregados (Aggregates)

DDD: São compostos por uma raiz (Aggregate Root) que controla a consistência das entidades e objetos de valor relacionados.

SOLID - SRP: A raiz do agregado é responsável por coordenar as operações no agregado.

SOLID - OCP (Open-Closed Principle): É possível estender a lógica do agregado sem modificar sua implementação, utilizando métodos adicionais ou serviços.

Exemplo:

```
public class Pedido
{
    private readonly List<ItemPedido> _itens = new();
    public IReadOnlyCollection<ItemPedido> Itens =>
        _itens.AsReadOnly();
    public void AdicionarItem(ItemPedido item) =>
        _itens.Add(item);
}
```

- Repositórios (Repositories)

DDD: Fornecem abstrações para persistir e recuperar agregados.

SOLID - DIP (Dependency Inversion Principle): A lógica de persistência depende de abstrações (interfaces), permitindo que a implementação concreta (e.g., EF Core) seja trocada sem impacto no domínio.

SOLID - ISP (Interface Segregation Principle): Interfaces específicas evitam dependências desnecessárias.

Exemplo:

```
public interface IPedidoRepository
{
    Task<Pedido> ObterPorIdAsync(Guid id);
}
```

```
Task SalvarAsync(Pedido pedido);  
}
```

- Serviços de Domínio

DDD: Contêm lógica de negócio que não pertence a uma entidade ou agregado.

SOLID - SRP: Cada serviço lida com uma lógica única e clara.

SOLID - OCP: Serviços podem ser estendidos sem modificar sua lógica básica.

Exemplo:

```
public class CalculoImpostoService  
{  
    public decimal Calcular(decimal valor) => valor * 0.1M;  
}
```

- Fábricas (Factories)

DDD: Gerenciam a criação de objetos complexos.

SOLID - SRP: Encapsulam a lógica de criação em um único ponto.

SOLID - DIP: Dependem de abstrações para criar instâncias.

Exemplo:

```
public class PedidoFactory  
{  
    public Pedido CriarPedido(Guid clienteId, List<ItemPedido>  
itens) => new Pedido(clienteId, itens);  
}
```

- Módulos

DDD: Agrupam conceitos relacionados para maior coesão.

SOLID - SRP e OCP: Cada módulo foca em um contexto específico e pode ser estendido separadamente.

- Eventos de Domínio

DDD: Capturam mudanças significativas no domínio.

SOLID - DIP: Eventos dependem de abstrações para serem publicados, permitindo o uso de diferentes implementações.

Exemplo:

```
public record PedidoCriado(Guid PedidoId, DateTime DataHora);
```

- **Contexto Delimitado (Bounded Context)**

DDD: Divide o sistema em áreas de responsabilidade claras.

SOLID - SRP e ISP: Cada contexto foca em resolver problemas específicos do negócio e mantém contratos bem definidos para comunicação.

- **Anti-Corrupção Layer (ACL)**

DDD: Protege o domínio de influências externas indesejadas.

SOLID - SRP e DIP: Centraliza a lógica de transformação e abstrai dependências externas para proteger o núcleo do sistema.

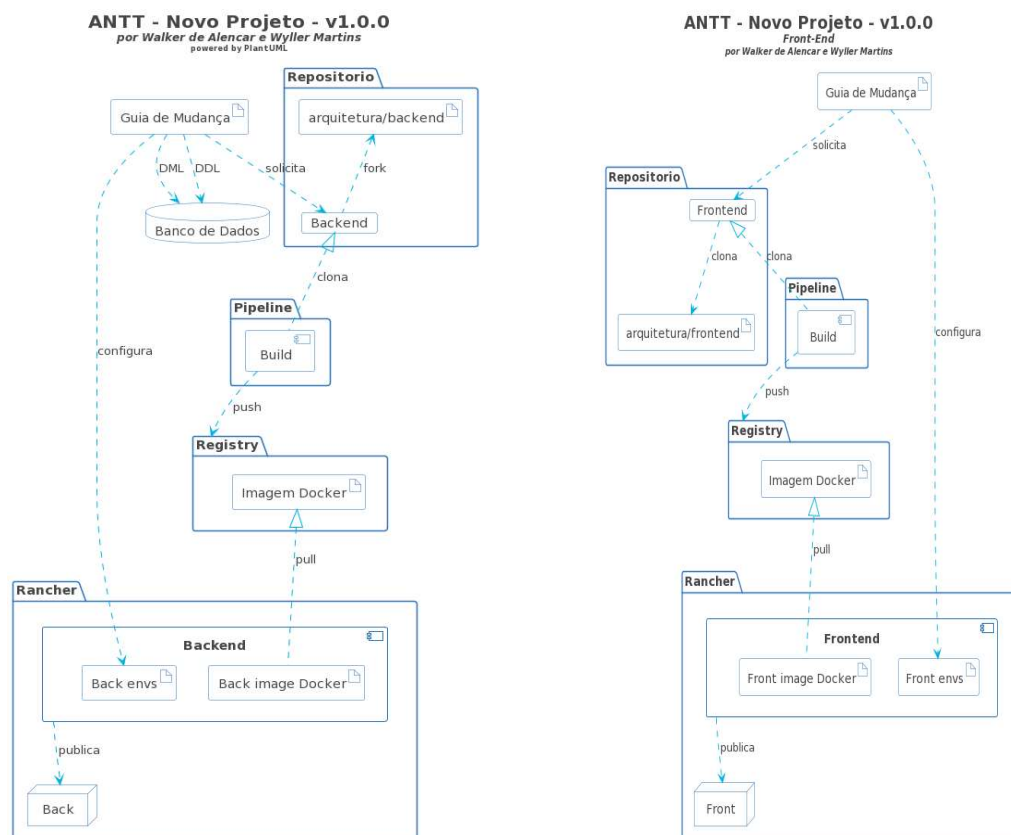
5 CI/CD – Integração Contínua e Entrega Contínua

5.1 Visão Macro do Fluxo CI/CD

CI/CD é um conjunto de práticas e princípios que automatizam o ciclo de vida de desenvolvimento de software, desde a escrita do código até sua implantação em produção, proporcionando ciclos de desenvolvimento mais rápidos, confiáveis e eficientes.

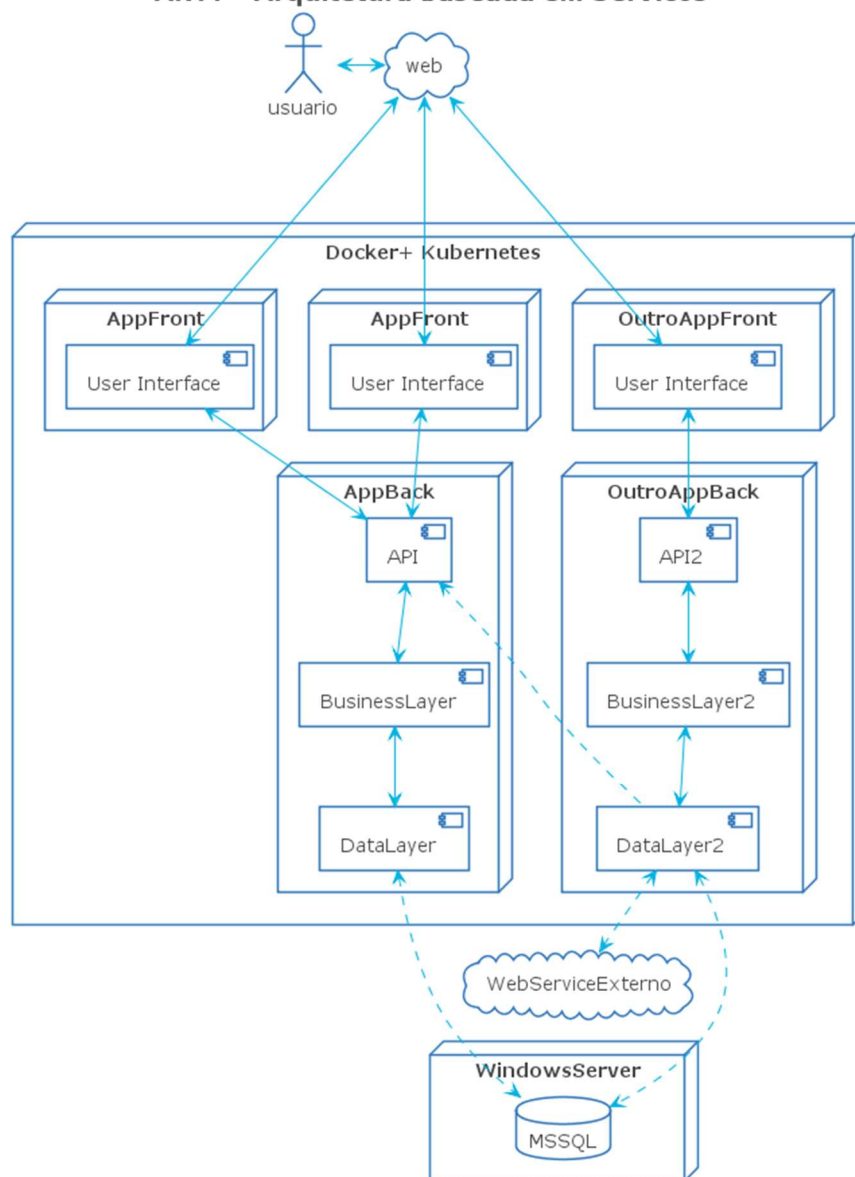
A imagem abaixo ilustra o fluxo de CI/CD implementado no ambiente da ANTT, ele contempla o cluster necessário para a aplicação.

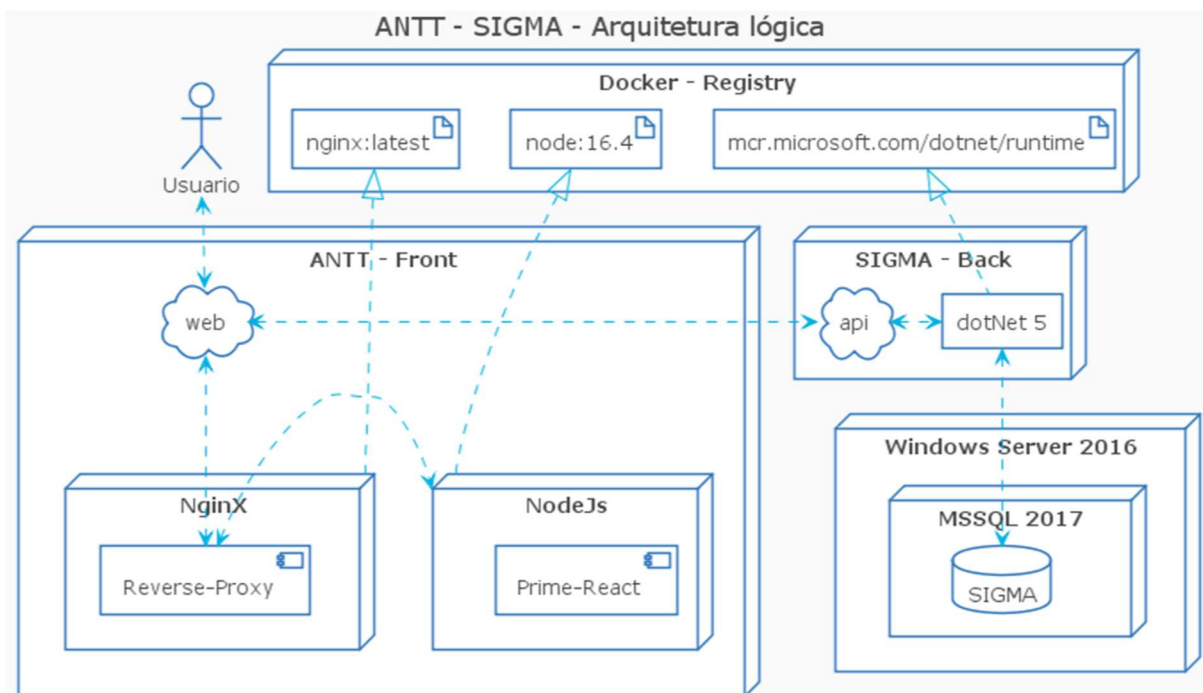
Representação Gráfica do Fluxo de Build e Implantação da Arquitetura



Essa divisão possibilita realizar manutenções de forma independente e distinta, bem como aumentar a capacidade de escala horizontal sob demanda. Por exemplo: pode-se colocar 3 instâncias de frontend trabalhando com 2 instâncias de backend, incrementando a disponibilidade do sistema.

ANTT - Arquitetura baseada em Serviços





5.2 Conceitos Principais do CI/CD

Integração Contínua (CI – Continuous Integration)

A Integração Contínua é a prática de integrar alterações de código de múltiplos desenvolvedores em um repositório compartilhado, várias vezes ao dia. Cada alteração acionará automaticamente um conjunto de testes e builds para validar a nova versão antes de ser mesclada ao código principal.

Benefícios da Integração Contínua:

- Detecta erros rapidamente, evitando falhas em produção.
- Reduz conflitos entre códigos de diferentes desenvolvedores.
- Automatiza testes e validações, garantindo qualidade do software.

Exemplo de fluxo de CI:

- Desenvolvedor realiza o commit no repositório (Git, GitLab, TFS).
- O pipeline de CI é acionado automaticamente.
- O código é analisado e compilado.
- Os testes automatizados (unitários, integração, segurança, etc.) são executados.

- Se tudo estiver correto, o código é mesclado no branch principal.

Implantação Contínua (CD – Continuous Deployment)

A Implantação Contínua é a evolução da Entrega Contínua, onde o código validado é implantado automaticamente em produção sem necessidade de aprovação manual.

Benefícios da Implantação Contínua:

- Redução do tempo entre o desenvolvimento e a entrega para usuários finais.
- Eliminação da necessidade de intervenção manual.
- Feedback rápido, permitindo ajustes ágeis e constantes.

Exemplo de fluxo de Continuous Deployment:

- Após aprovação nos testes, a nova versão é automaticamente implantada em produção.
- Monitoramento contínuo verifica se a aplicação está funcionando corretamente.
- Caso ocorram falhas, o rollback (reversão automática) pode ser acionado.

5.3 Componentes Essenciais de um Pipeline CI/CD

Versionamento de Código (SCM - Source Control Management)

Ferramentas como GitLab e TFS são usadas para gerenciar o código-fonte e suas versões. Na ANTT, o padrão é o GitLab e o Git Desktop com cliente de acesso aos servidores de versionamento.

Automação do Build

A construção do código deve ser automatizada usando ferramentas como:

- Nuget (.NET) – padrão na ANTT;

- Yarn/NPM (Node.js, React, Vue) – padrão na ANTT;
- Webpack (Frontend);
- Maven (Java);
- Gradle (Java, Kotlin).

Testes Automatizados

Os pipelines de CI/CD devem conter testes para garantir a qualidade do código:

- Testes Unitários: Verificam componentes individuais (ex: Jest, JUnit, xUnit).
- Testes de Integração: Validam interações entre componentes.
- Testes de Segurança: Identificam vulnerabilidades (ex: SonarQube, OWASP ZAP).
- Testes de Carga e Performance: Avaliam desempenho sob estresse (ex: JMeter, Locust).

Construção e Distribuição de Imagens Docker

No CI/CD implementado pela ANTT, aplicações são empacotadas como imagens Docker para garantir portabilidade e padronização. Exemplos de registries para armazenar imagens:

- Harbor (registro privado).

Implantação com Kubernetes e Rancher

A entrega das aplicações deve ser feita com Kubernetes, onde os Pods são gerenciados automaticamente. O Rancher é a aplicação responsável pela administração de clusters Kubernetes.

A adoção do CI/CD é essencial para empresas que buscam aumentar a eficiência no desenvolvimento de software, reduzir erros e acelerar o tempo de entrega. Seja com Integração Contínua (CI), Entrega Contínua (CD) ou Implantação

Contínua (CD), cada etapa contribui para um fluxo de desenvolvimento mais ágil e confiável.

6 Dados

6.1 Considerações de Modelagem Específicas

O ciclo de vida de um dado operacional compreende todas as etapas pelas quais um dado transita dentro de um sistema, desde o seu surgimento até o momento em que deixa de ser atualizado e passa a ser utilizado somente para análises históricas. Em outras palavras, é o fluxo completo pelo qual o dado é criado, processado, modificado, e, por fim, arquivado ou migrado para ambientes analíticos. Seguem os principais pontos desse conceito:

- **Criação:** Nesta fase, o dado é gerado a partir de uma transação ou evento do sistema. É o ponto inicial onde o dado operacional nasce, seja através de entrada manual, automática, integração com outros sistemas ou dispositivos.
- **Processamento e Atualização:** Durante seu período ativo, o dado é utilizado em operações diárias e pode ser atualizado ou transformado conforme as regras de negócio. Aqui, o dado passa por verificações, validações e eventuais transformações necessárias para atender aos processos operacionais.
- **Utilização Operacional:** Enquanto o dado está sendo atualizado e utilizado, ele serve para suportar as operações transacionais do sistema, respondendo a demandas em tempo real, como consultas, inserções e modificações que garantem a continuidade dos processos do negócio.
- **Encerramento do Ciclo:** Quando o dado atinge o fim do seu ciclo operacional—ou seja, quando não há mais necessidade de atualizá-lo para suportar as operações em tempo real—ele é considerado “concluído” para fins operacionais. Neste estágio, o dado pode ser migrado ou arquivado, sendo mantido para análises históricas,

geração de relatórios, ou para a realização de estudos e insights que suportem decisões estratégicas.

A definição de modelo de dados e arquitetura de solução para lidar com dados operacionais. Dentro do modelo de dados de cada solução devem ser identificados os dados operacionais, que têm um ciclo de vida definido e os dados de domínio, os quais possuem pouquíssimas mudanças e nenhum ciclo de vida, ou seja, quase dados estáticos.

Abaixo segue a definição, mais detalhada, de dados operacionais e de domínio:

- **Dados Operacionais:** São os dados que representam o ciclo ativo de processamento do sistema. São criados, atualizados e utilizados em tempo real para operações e, ao final de seu ciclo de vida (quando não há mais necessidade de operações transacionais), são destinados para análise histórica.
- **Dados de Domínio:** São dados que definem características e regras do negócio (exemplo: tabelas de configuração, catálogos, códigos de status). Eles sofrem poucas alterações e não têm um ciclo de vida transacional relevante.

Uma vez identificado as entidades operacionais, a arquitetura de solução, através do modelo de dados, deve atender aos requisitos:

- Identificar, mapear, e descrever as regras de negócio, com seus estados dentro do ciclo de vida do dado operacional;
- Atender a alta demanda de escrita/leitura para dados, de forma performática com poucos dados até grandes volumes de dados;
- Manter integridade e performance, garantindo que as operações transacionais não sejam afetadas pelos processos de pesquisa de dados;
- Permitir a migração e consolidação dos dados operacionais, após seu ciclo de vida, para um ambiente de consulta analítica, de forma simples e segura, sem perda de dados

Na criação do modelo de dados para atender ao negócio devem ser identificadas as entidades de dados operacionais, ou seja, que tem um ciclo de vida

da informação. Em toda entidade operacional devem ser definidos campos para seu particionamento horizontal pelo Sistema de Gerenciamento de Banco de Dados.

6.2 Relacionamentos e Integridade Referencial

Relacionamentos 1:N e N:N: Definir claramente os relacionamentos entre entidades. Por exemplo, uma entidade “Transação” pode relacionar-se com várias “Entradas de Pagamento”, mas cada “Pagamento” deve referenciar apenas uma “Transação”. Em casos de relações muitos-para-muitos (N:N), deve-se utilizar tabelas de associação que mantenham os vínculos e a integridade dos dados.

Chaves Estrangeiras e Constraints: Todas as relações devem ser reforçadas por constraints no banco de dados, de modo que a remoção ou atualização de um registro em uma entidade não quebre a integridade dos dados em entidades relacionadas.

6.3 Técnicas de Otimização para Altos Volumes

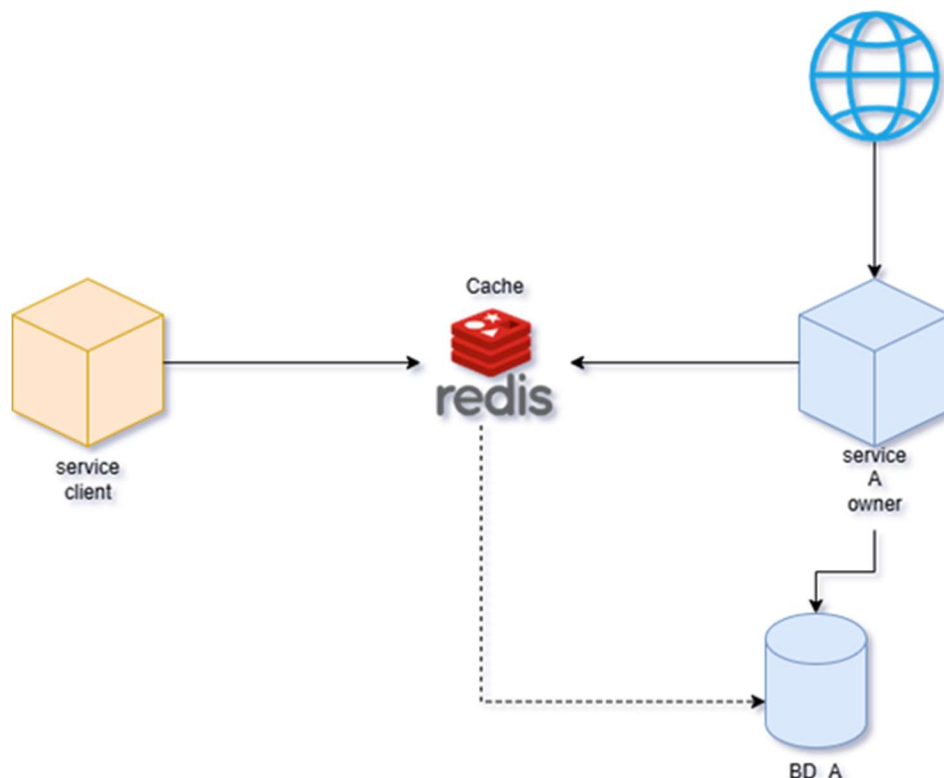
Particionamento e Arquivamento: Deve-se estruturar as tabelas de forma a facilitar a migração de dados que atingiram o final do ciclo operacional para ambientes de análise, sem impactar a performance do sistema operacional.

Índices e Materialized Views: Convém utilizar índices específicos e, quando aplicável, visões materializadas para acelerar consultas frequentes e agregações, sem comprometer a atualização dos dados operacionais.

6.4 Volumetria

Cache de 2º Nível: As aplicações backend/API devem implementar cache de 2º nível para os dados de domínio, ou mesmo os dados que têm baixo número de alteração como, por exemplo, um dado que pode ser alterado até 2 vezes em uma semana. A implementação de cache de 2º nível evitará um consumo excessivo de requisições a base de dados. Desta forma a aplicação fica mais leve e poderá processar com um desempenho satisfatório.

Grande Volume de Consumo: As aplicações/serviços que precisarem consumir outro serviço, devem identificar a quantidade volumétrica a ser consumida para que em casos de alto volume como, por exemplo, mais de 60 requisições por segundo, um mapeamento dos dados necessários deve ser feito e encaminhado para um servidor de cache. Identificada a alta volumetria de consumo do serviço, este deve ser isolado utilizando um servidor de cache, como o REDIS.



Objetivos do Cache Redis:

- **Redução da Latência:** Armazenar os dados mais acessados ou os resultados de consultas frequentes para que possam ser recuperados de forma rápida, sem a necessidade de consultar o banco de dados principal.
- **Alívio da Carga do Banco de Dados:** Diminuir o número de requisições diretas ao banco de dados, melhorando a performance geral do sistema, principalmente em cenários de alto volume de requisições.

Requisitos Funcionais Específicos do Redis:

- Atualização em Tempo Real: O servidor de cache deve ser atualizado em intervalos definidos para refletir mudanças importantes nos dados operacionais.
- Invalidation e Expiração: Implementar políticas de expiração e invalidação do cache para garantir que os dados armazenados estejam sempre atualizados conforme o ciclo de vida operacional. Por exemplo, ao final do ciclo de um dado (quando o status passa para “finalizado”), o cache deve ser invalidado ou atualizado para refletir essa mudança.
- Alta Disponibilidade: O ambiente Redis deve ser configurado em modo de cluster ou replicação para garantir alta disponibilidade e tolerância a falhas.

Integração com o Modelo de Dados:

- Cache de Consultas Frequentes: As consultas que agregam dados operacionais (como contagens, agregações de status ou buscas por dados críticos) serão direcionadas ao Redis para respostas rápidas.
- Sincronização com o Banco de Dados: Estabelecer mecanismos de sincronização que garantam que, sempre que os dados operacionais sofrerem alterações significativas, o cache Redis seja atualizado ou invalidado para manter a consistência.
- Arquitetura de Microsserviços: Os serviços que consomem dados do cache (por exemplo, APIs de alta performance, dashboards em tempo real) deverão ser projetados para utilizar o Redis como camada intermediária, garantindo que as requisições de alto volume sejam tratadas com baixa latência.

6.5 Internalização e Hospedagem

Para os casos em que a solução de TIC não for desenvolvida inicialmente pela SUTEC, existe a possibilidade de Internalização ou Hospedagem. A hospedagem pode ser feita em servidores físicos ou virtuais disponíveis no ambiente Datacenter da ANTT, ou pode ser oferecida por um provedor de serviços da cloud, caso haja contrato

vigente. Já a Internalização refere-se ao processo cujas soluções tecnológicas são mantidas e potencialmente evoluídas por equipes da SUTEC.

Na hospedagem de sistemas desenvolvidos por outros órgãos, entidades ou unidades da ANTT (externas à SUTEC) que não forem internalizados pela área de TIC desta Agência, as manutenções corretivas/evolutivas ficarão sob responsabilidade daqueles que o tiverem desenvolvido.

Futuramente, caso haja o interesse pela internalização, é necessário que o sistema esteja adequado aos padrões da SUTEC.

Na implantação de sistemas desenvolvidos por outros órgãos/entidades é imprescindível que seja verificada a sua compatibilidade com o padrão arquitetural de sistemas da ANTT e com a capacidade computacional disponível.

Caso haja a oportunidade de implementação com base nas tecnologias adotadas e licenciadas pela ANTT, esta opção deve ser priorizada. O mesmo critério deve ser adotado no caso de contratos de concessão que preveem a disponibilização de sistemas e/ou APIs para interoperabilidade.

Caso a solução já esteja pronta para produção, é necessário que a SUTEC realize uma análise de viabilidade da sua implantação, avaliando os riscos envolvidos do ponto de vista técnico e seus impactos nos custos, escopo e cronograma do projeto e formalize via parecer técnico a viabilidade de internalização da solução.

7 Referências

Redis - <https://redis.io/>

ePING - <https://eping.governoeletronico.gov.br/>

Guias de requisitos mínimos de segurança:

https://www.gov.br/governodigital/pt-br/privacidade-e-seguranca/ppsi/guia_requisitos_minimos_apis.pdf

https://www.gov.br/governodigital/pt-br/privacidade-e-seguranca/ppsi/guia_requisitos_minimos_web.pdf

https://www.gov.br/governodigital/pt-br/privacidade-e-seguranca/ppsi/guia_requisitos_minimos_apps.pdf

LGPD - https://www.gov.br/governodigital/pt-br/privacidade-e-seguranca/guias/guia_lgpd.pdf

eMAG - <https://emag.governoeletronico.gov.br>

Padrão Digital de Governo - <https://www.gov.br/ds/home>

Rancher - <https://rancher.antt.gov.br>

Harbor - <https://registry.antt.gov.br>

Suporte linguagem e plataforma - <https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/language-versioning>

<https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/configure-language-version>

Primereact - <https://primereact.org/>

StyledComponents - <https://styled-components.com/docs>

Redux - <https://redux.js.org/>

Normas e Padrões de Administração de Banco de Dados ANTT - <https://gitlab.antt.gov.br/antt/arquiteturas/docs-arquiteturas/-/blob/master/banco-dados/Normas%20e%20Padroes%20de%20Administra%C3%A7%C3%A3o%20de%20Banco%20de%20Dados%20da%20ANTT%20v1.7-2.docx>

DDD - <https://engsoftmoderna.info/artigos/ddd.html>

<https://www.domainlanguage.com/ddd/>

Guia de Boas Práticas – LGPD - https://www.gov.br/governodigital/pt-br/privacidade-e-seguranca/guias/guia_lgpd.pdf

Markdown - <https://www.markdownguide.org/extended-syntax/>