Regras de Boas Práticas de Codificação do Grupo

0. Como aplicar estas regras

Estas regras devem ser seguidas em todo código desenvolvido para o projeto. Em partes já existentes, sempre que houver manutenção ou evolução, o trecho alterado deve ser atualizado para ficar alinhado a este padrão.

1. Padrão de notação e formatação

Para todo mundo conseguir ler o código da mesma forma, vamos seguir este padrão:

- Formatação
    - Identação fixa (por exemplo, 4 espaços) em todos os arquivos.
    - Um comando por linha, evitando linhas muito longas.
    - Sempre usar espaços em torno de operadores (a + b, não a+b).
    - Usar espaço após vírgulas, como em func(a, b, c).
- Nomes
    - Classes: **lowerCamelCase** (ex.: usuarioService, pedidoController).
    - Variáveis e métodos: **lowerCamelCase** (ex.: dataNascimento, calcularTotal).
    - Evitar nomes genéricos como x, y, temp para elementos de regra de negócio.
    - Usar nomes que expressam claramente o propósito (ex.: dataCompra, valorTotalPedido).
- Estruturas de controle
    - Sempre usar chaves { } em if, else, for, while, mesmo com uma linha só, para evitar erros de manutenção.

2. Comentários e documentação mínimos

A ideia é que qualquer pessoa do grupo consiga entender o código sem dificuldade excessiva.

- Documentação de funções
    - Funções públicas ou mais importantes devem ter uma breve documentação indicando: Propósito da função; Parâmetros principais; O que ela retorna (quando houver).
- Comentários no meio do código
    - Comentar o porquê de algo ser feito, não apenas o que está sendo feito.
    - Se for necessário comentar demais um trecho para que ele seja entendido, avaliar e extrair a lógica para um método com nome claro.

- Manutenção
    - Sempre que o código for alterado, atualizar os comentários relacionados. Comentários desatualizados são piores do que a ausência de comentários.

3. Nomes significativos e responsabilidade única (SRP – SOLID)

Essas regras ajudam a evitar que a regra de negócio fique dispersa e difícil de entender.

- Nomes significativos
    - Classes: usar substantivos (ex.: NotaFiscal, PedidoService).
    - Métodos: usar verbos (ex.: criarUsuario, validarDatas).
- Responsabilidade única (SRP na prática)
    - Cada classe deve ter um papel principal bem definido (ex.: UsuarioController lida com requisições e respostas; UsuarioService lida com regras de negócio relacionadas ao usuário).
    - Em código novo, seguir essas práticas desde o início. Em código antigo, aplicar as melhorias quando for necessário alterar o trecho.

4. Funções simples e sem repetição desnecessária (Clean Code)

O objetivo é evitar complexidade desnecessária e facilitar a leitura e manutenção.

- Preferir soluções simples em vez de soluções excessivamente complexas.
- Evitar copiar e colar a mesma lógica em vários lugares; quando isso acontecer, extrair para um método reutilizável.
- Evitar condicionais muito aninhadas (ifs dentro de ifs). Quando possível, usar retornos antecipados (early return) ou quebrar a lógica em métodos auxiliares.

5. Testes e tratamento de erros

Esta regra existe para garantir o mínimo de segurança sem tornar o processo pesado demais.

- Testes
    - Funcionalidades importantes e essenciais para o sistema devem ter pelo menos um teste automatizado (unitário ou de integração).
    - Antes de integrar mudanças na branch principal, executar os testes existentes.
- Erros
    - Preferir lançar exceções com mensagens claras em vez de códigos numéricos sem significado.
    - Validar entradas críticas, como valores nulos, formatos inválidos ou dados fora de faixa esperada.

- Garantir que mensagens de erro retornadas a usuários ou outros sistemas sejam compreensíveis.

6. Regra de ouro

Em caso de dúvida, priorizar código simples, legível e consistente com o padrão já adotado no projeto.