Boas Práticas de Programação adotadas no Projeto

Este projeto adota um conjunto de boas práticas de programação simples, que têm como objetivo garantir a qualidade, clareza, manutenibilidade e consistência do código-fonte. As diretrizes descritas abaixo orientam a implementação de todas as funcionalidades, assegurando que diferentes desenvolvedores possam contribuir de maneira padronizada, reduzindo erros e facilitando a evolução do sistema.

1. Comentários

Os comentários utilizados no código devem ser:

- curtos e objetivos;
- direcionados a esclarecer a finalidade de partes do código e auxiliar na separação de blocos lógicos.

O objetivo é evitar comentários desnecessários e manter o código limpo, mas ainda assim compreensível quando há trechos mais complexos.

2. Padrões de notação

Para preservar clareza e uniformidade, adotamos padrões específicos para nomes de funções, variáveis e módulos:

- Funções e variáveis devem ter nomes intuitivos, adequados ao comportamento que representam.
- Nomes de funções devem começar com verbos (*create*, *get*, *update*, *find*, *delete*, *cancel*, etc.).

Essa convenção facilita a leitura do código e mantém previsibilidade nas ações implementadas.

3. Indentação Padronizada

Todo o código deve seguir um padrão único de indentação, de modo a facilitar a leitura do código e a identificação de blocos lógicos.

O uso de ferramentas automáticas de formatação (como Prettier ou ESLint) é recomendado para garantir consistência em todos os arquivos.

4. Tratamento de Exceções com try/catch

Todas as operações suscetíveis a falhas devem estar protegidas por estruturas adequadas de tratamento de exceção:

- uso sistemático de **try/catch**;
- mensagens de erro claras e úteis;
- tratamento coerente com o contexto de negócio;
- nunca deixar exceções silenciosas.

Esse cuidado aumenta a robustez do sistema, evita que falhas interrompam o fluxo de execução e facilita o diagnóstico de problemas.

5. Evitar negação em condições de if

Para reduzir ambiguidades e melhorar a legibilidade, recomenda-se evitar estruturas condicionais com negação (ex.: `if (!condicao)`), substituindo-as por versões afirmativas ou por nomes de variáveis mais claros.

6. Aplicação do princípio DRY (Don't Repeat Yourself)

O código deve evitar duplicações desnecessárias, consolidando comportamentos repetidos em:

- funções reutilizáveis,
- serviços independentes,
- módulos compartilhados.

Tal prática é necessária para auxiliar na manutenção do código, aumentar a coerência do sistema e facilitar alterações futuras.