Manual de Boas Práticas de Codificação

Projeto Trio Bit Garage

Equipe:
Guilherme Lirio Miranda
Fábio Damas Valim
Caio Finnochio Martins

Lavras - MG
30 de novembro de 2025

1 Introdução

Este documento estabelece as diretrizes de desenvolvimento para o projeto *Trio Bit Garage*. O objetivo é garantir a legibilidade, manutenibilidade e escalabilidade do código, seguindo princípios de *Clean Code* e padrões de arquitetura MVC. A adesão a estas 6 regras é obrigatória para todos os membros da equipe.

2 Regra 1: Padrão de Notação (Naming Conventions)

Para manter a consistência entre o Frontend e o Backend (Node.js), adotamos as seguintes convenções de nomenclatura:

- **Classes e Models:** Utilizar **PascalCase**.
  - Exemplo: `Veiculo`, `LocacaoController`, `AuthService`.
- **Variáveis, Funções e Métodos:** Utilizar **camelCase**. Nomes devem ser descritivos (substantivos para variáveis, verbos para funções).
  - Exemplo: `const novoUsuario`, `function calcularTotal()`.
- **Banco de Dados (Colunas):** Conforme configuração do Sequelize (`underscored: true`), as colunas no banco utilizam **snake_case**, mas são mapeadas para camelCase no JavaScript.
  - Exemplo: Tabela `data_nascimento` → Objeto JS `dataNascimento`.
- **Constantes:** Utilizar **UPPER_CASE_SNAKE_CASE**.
  - Exemplo: `MAX_RETRY_ATTEMPTS = 5`.

3 Regra 2: Documentação e Comentários

O código deve ser autoexplicativo sempre que possível. No entanto, a documentação é obrigatória em camadas de integração:

- **JSDoc:** Obrigatório no topo de cada *Service* e *Controller* para descrever o propósito do módulo.
- **Comentários de Regra de Negócio:** Obrigatório comentar trechos onde uma regra de negócio complexa é aplicada (ex: validação de disponibilidade de veículo).
- **Evitar:** Comentários óbvios (ex: `// Cria variável`).

```
/**
 * Service responsavel pela regra de negocio de Locacoes.
 * Verifica disponibilidade e atualiza status do veiculo.
 */
const LocacaoService = {
  create: async (data) => {
```

```
    // REGRA DE NEGOCIO: Verifica se o carro esta disponivel antes de alugar
    if (veiculo.situacao !== 'DISPONIVEL') {
      throw new Error('Veiculo indisponivel.');
    }
    // ...
  }
};
```

Listing 1: Exemplo de documentação no LocacaoService

4 Regra 3: Arquitetura e Responsabilidade Única (SRP)

Seguindo o princípio SOLID de Responsabilidade Única (SRP) e a arquitetura MVC:

- **Controllers:** Devem apenas receber a requisição HTTP (req), validar parâmetros básicos e devolver a resposta (res). Não devem conter lógica de negócio.
- **Services:** Contêm toda a lógica de negócio ("o coração do sistema"). É aqui que ocorrem as validações, cálculos e chamadas ao banco.
- **Models:** Definem apenas a estrutura dos dados e relacionamentos.

5 Regra 4: Tratamento de Erros e Async/Await

Para evitar o "Callback Hell"e garantir que a aplicação não quebre inesperadamente:

- É obrigatório o uso de **async/await** para operações assíncronas (banco de dados).
- Todo método de *Controller* deve envolver a chamada do serviço em um bloco **try/-catch**.
- O bloco `catch` deve retornar um status HTTP apropriado (ex: 400 para erro de validação, 500 para erro interno).

```
async create(req, res) {
  try {
    const result = await Service.create(req.body);
    return res.status(201).json(result);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
}
```

Listing 2: Padrao Try-Catch no Controller

6 Regra 5: Formatação e Estilo (Clean Code)

Para garantir a legibilidade visual do código entre diferentes desenvolvedores:

- **Indentação:** 2 espaços (padrão comum em projetos JavaScript/Node modernos).
- **Ponto e vírgula:** Obrigatório o uso de ponto e vírgula (`;`) ao final das instruções para evitar erros de interpretação do motor V8.
- **Aspas:** Preferência por aspas simples (`'string'`) para strings normais e Template Literals (`` `texto $var` ``) para concatenação.

7 Regra 6: Clareza e Limpeza (DRY - Don't Repeat Yourself)

- **Evitar Código Morto:** Remover imports não utilizados, variáveis declaradas e não usadas e trechos de código comentados antigos.
- **Reutilização:** Se uma lógica é usada em mais de um lugar (ex: verificar se usuário é Admin), ela deve ser extraída para uma função auxiliar ou Middleware, nunca duplicada.
- **Inglês vs Português:** Mantenha consistência. Se o banco de dados está em Português (`data_retirada`), o código deve seguir o padrão ou fazer a tradução consistente nos DTOs. No Trio Bit Garage, optamos por manter os termos do domínio em Português para facilitar o alinhamento com os Requisitos.