Guia de Implantação

Versão 1.1

F360 – Finance360

Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 16/11/2024 | 1.0 | Criação do documento | Guilherme Fagundes de Almeida |
| 11/12/2024 | 1.1 | Atualização das partes comentário, nomeação e declaração. | Guilherme Fagundes de Almeida |
|  |  |  |  |

SUMÁRIO

[1. Introdução 4](#_Toc182766372)

[1.1. Referências 4](#_Toc182766373)

[2. ORGANIZAÇÃO E ESTILO DO CÓDIGO 4](#_Toc182766374)

[2.1. Comentarios 4](#_Toc182766375)

[2.2. Nomeação 5](#_Toc182766376)

[2.3. Declaração 5](#_Toc182766377)

[2.4. Reutilização 5](#_Toc182766378)

[2.5. Tratamento de erros e exceções 6](#_Toc182766379)

[3. Diretrizes Gerais 6](#_Toc182766380)

# Introdução

O **Finance360** é uma plataforma digital para gerenciar finanças pessoais e de pequenas empresas, focada em acessibilidade, escalabilidade e segurança. Este guia de implementação ajudará a garantir que o código seja limpo, eficiente e sustentável ao longo do tempo. O projeto envolve diversas tecnologias como Node.js, Express, SQLite, AWS, bcrypt, e integrações com APIs externas como a BRAPI.

Este guia está dividido em várias seções que cobrem práticas de codificação, cronograma de desenvolvimento, tratamento de erros e exceções, e diretrizes gerais.

## Referências

* Guia - Padroes e Ferramentas para Projeto de Banco de Dados.pdf

# ORGANIZAÇÃO E ESTILO DO CÓDIGO

Para garantir que o código seja limpo, compreensível e fácil de manter, o projeto seguirá algumas convenções de estilo e organização:

* **Indentação**: Utilize 2 espaços para indentação.
* **Espaços em branco**: Use espaços em branco ao redor de operadores e após vírgulas para melhorar a legibilidade.
* **Max Linha**: Linhas de código devem ter no máximo 100 caracteres.
* **Quebra de linha**: Sempre que uma função ou expressão for muito longa, quebre-a para melhorar a legibilidade.
* **Tamanho de classe**: Não mais que 600 linhas.
* **Tamanho de método**: Não mais que 40 linhas.

## Comentarios

Comentários devem ser usados com moderação, apenas quando o código não for imediatamente claro ou para explicar a lógica de decisões complexas.

* **Comentário de bloco** (para descrever seções de código ou lógica complexa):

// Lógica de autenticação de usuário

static async authenticateUser(email, password) {

const user = await UserDAO.findUserByEmail(email);

if (!user) {

throw new Error('Usuário não encontrado.');

...

## Nomeação

* Arquivo JS (Pacote model):

· NomeDaClasse.js

* Arquivo JS (Pacote controllers):

· ClasseController.js

* Arquivo JS (Pacote services):

· ClasseService.js

* Arquivo JS (Pacote models):

· Classe.js

## Declaração

* Deixe as funções bem definidas, com parâmetros claros e retornos explícitos:

// Método para validar senha

static async validatePassword(password, hash) {

return await bcrypt.compare(password, hash);

}

}

## Reutilização

* Aplicar os padrões de atribuição de responsabilidade nas classes (ver padrões GRASP).

## Tratamento de erros e exceções

* Garantir que as exceções subam para a camada de apresentação, usando a clausula catch nos métodos envolvidos.

# Diretrizes Gerais

* **Desenvolvimento Ágil**: Adote práticas de desenvolvimento ágil, como commits frequentes e integração contínua.
* **Versionamento**: Utilize o **Git** para versionamento do código, fazendo commits significativos e com mensagens claras.
* **Documentação**: Documente o código e a API. Use ferramentas como Swagger ou JSDoc para documentar suas rotas e funções.