Guia de Implantação

Versão 0.1

F360 – Finance360

Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 16/11/2024 | 0.1 | Criação do documento | Guilherme Fagundes de Almeida |
|  |  |  |  |
|  |  |  |  |

SUMÁRIO

[1. Introdução 4](#_Toc182765714)

[1.1. Referências 4](#_Toc182765715)

[2. ORGANIZAÇÃO E ESTILO DO CÓDIGO 4](#_Toc182765716)

[2.1. Comentarios 4](#_Toc182765717)

[2.2. Nomeação 5](#_Toc182765718)

[2.3. Declaração 5](#_Toc182765719)

[2.4. reutilização 5](#_Toc182765720)

[2.5. Tratamento de erros e exceções 6](#_Toc182765721)

[3. Diretrizes Gerais 6](#_Toc182765722)

# 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 - Oracle Java Code Conventions.pdf;
* 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 80 caracteres.
* **Quebra de linha**: Sempre que uma função ou expressão for muito longa, quebre-a para melhorar a legibilidade.
* **Funções pequenas e focadas**: As funções devem realizar uma única tarefa. Caso seja necessário, crie funções auxiliares.
* **Tamanho de classe**: Não mais que 600 linhas.
* **Tamanho de método**: Não mais que 20 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):

/\* Esta função calcula o preço unitário médio de um investimento. \*/

function calcularPrecoUnitario(investimentos) {

// Lógica do cálculo...

}

* **Comentários de TODO**(comentários de alterações a serem realizadas posteriormente):

// TODO: Refatorar essa função para reduzir a duplicação de código.

## Nomeação

* Arquivo Java (Pacote model):

· NomeDaClasse.java

* Arquivo Java (Pacote controller):

· ClasseController.java

* Arquivo Java (Pacote service):

· ClasseService.java

* Arquivo Java (Pacote repository):

· ClasseRepository.java

## Declaração

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

function calcularLucro(valorInvestido, valorAtual) {

return valorAtual - valorInvestido;

}

Padrão de indentação de declaração de métodos:

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

## 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 Trows nos métodos envolvidos.

try {

Scanner scanner = new Scanner(System.in);

System.out.print("Digite o valor do investimento (mínimo R$ " + VALOR\_MINIMO\_INVESTIMENTO + "): ");

double valorInvestimento = scanner.nextDouble();

if (valorInvestimento < VALOR\_MINIMO\_INVESTIMENTO) { throw new Exception("Erro: O valor do investimento deve ser no mínimo R$ " + VALOR\_MINIMO\_INVESTIMENTO);

}

# 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.
* **Testes**: Crie testes automatizados usando frameworks como Jest ou Mocha para garantir que cada funcionalidade funcione corretamente.