**Guia de Implementação**

SUMÁRIO

[1. Introdução 2](#_Toc485801465)

[1.1. Referências 2](#_Toc485801466)

[2. Organização e Estilo do Código 2](#_Toc485801467)

[3. Comentários 2](#_Toc485801468)

[4. Nomeação 2](#_Toc485801469)

[5. Declaração 3](#_Toc485801470)

[6. Reutilização 3](#_Toc485801471)

[7. Tratamento de Erros e de Exceções 3](#_Toc485801472)

[8. teste unitário 3](#_Toc485801473)

[9. Diretrizes Gerais 4](#_Toc485801474)

# Introdução

Este documento tem por finalidade definir e padronizar as tecnicas, processos e métodos da produção do aplicativo “AgendaAI”. O projeto está fundamentado sobre a linguagem java e o framework springboot. Sendo assim, quaisquer padrões da linguagem e da tecnologia podem e devem ser aplicados durante a manutenção do código. Não somente o código, mas também os “commits” ao github também devem seguir padrões. A lista com toda a convenção usada será colocada na lista de referências, mas os mais importantes são, “docs:” para quaisquer alterações na documentação presente no repositório, “feat:” para qualquer “feature” adicionada, isto é qualquer alteração que adicione uma funcionalidade a aplicação como um todo ou ao código, “fix:” para qualquer alteração que apenas remova um bug ou conserte determinado trecho do código. Como o aplicativo não é um monolith e sim uma serie de microserviços que se integram, cada serviço deve ter apenas código que diga a respeito delas mesmo, a integração das API’s não é responsabilidade delas. A aplicação deve ser implementada no server-side.

## Referências

* Guia - https://www.oracle.com/java/technologies/javase/codeconventions-contents.html;
* Guia - https://www.conventionalcommits.org/pt-br/v1.0.0-beta.4/

# Organização e Estilo do Código

Contanto que a classe se mantenha fiel ao princípio de representar ou ter apenas uma única responsabilidade. O mesmo equivale para funções, embora não tenham tamanho fixo, uma função deve exercer apenas uma responsabilidade, qualquer outra responsabilidade deve ser abstraída em outro método e por aí em diante.

Classes muito grandes que retornam mais de um valor devem usar o estilo “return early”.

# Comentários

Comentarios devem ser adicionados apenas em métodos ou classes que não sejam autoexplicativas, e devem ser feitas como um comentario curto de não mais que 5 linhas.

Comentarios sobre alterações que devem ser feitas, tem que estar com a frase “TODO:” antes e não podem ter mais de 5 linhas.

# Nomeação

Arquivo Java

* NomeClasse.java.

# Declaração

Métodos devem seguir o padrão camel case.

1. Padrão de indentação de declaração de métodos:

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

# Reutilização

* Todo código deve ser dividido nos padrões do SpringBoot.
* Aplicar os padrões de atribuição de responsabilidade nas classes (ver padrões SOLID).

# Tratamento de Erros e de Exceções

* Se o retorno for um array ou json, retorne uma lista vazia, e não um erro.
* Qualquer método que em determinado ponto devolva null, deve ser interpretado como um erro.

# teste unitário

Não são utilizados testes de JUNIT, qualquer teste deve ser implementado com o github actions.

# Diretrizes Gerais

O arquivo README.MD do projeto deve estar sempre atualizado com instruções claras de como rodar o servidor e com as principais informações. O projeto deve ser feito sempre em duas branchs, sendo uma para o desenvolvimento rotineiro, e uma vez testado e aprovado, colocado na main para ser executado em produção.