

---

Além dos comentários

*/\*\* Criando Documentação de Valor em PHP \*/*

João Vitor Santos



# Quem sou eu?

---

“

## Carreira e Formação

- </> Desenvolvedor PHP na IXCSOft
- 🎓 Técnico em Redes de Computadores
- 🎓 Graduado em Análise e Des. de Sistemas
- 🎓 Atualmente cursando Ciência da Computação

## Fora do código

- ♟ Mestre de Xadrez online (meu rating não sai de 600)
- 🎮 Vibe-codo Apps (in)úteis nas horas vagas
- 🐾 Pai de pet

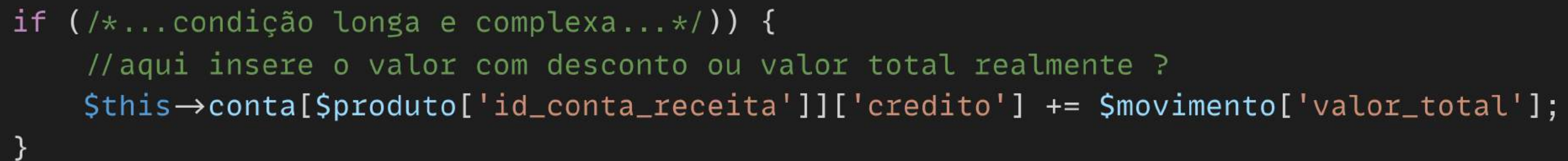




## O Problema: Quando o comentário atrapalha

---

```
1 if ($cm[$ib]) > ($vl[$ib])) { //se o custo médio é maior que o valor de venda do item $this->dados['controla_estoque']
2     $v = $cm[$ib] - $vl[$ib]; // diferença custo médio - valor de venda
3     //contabiliza a débito a diferença na conta de despesa do grupo de produtos.
4 }
```



```
if (/*...condição longa e complexa...*/) {
    //aqui insere o valor com desconto ou valor total realmente ?
    $this->conta[$produto['id_conta_receita']]['credito'] += $movimento['valor_total'];
}
```






# Não comunicar é o problema

---

Comentário não é o real problema

Sem uma boa documentação:

-  Onboarding
-  Medo
-  Tempo perdido

*"Código bom não é só o que a máquina entende, mas o que outros humanos conseguem entender."*



# O Mito do "Código Autoexplicativo"

O código mostra O QUÊ, mas raramente explica O PORQUÊ.

O QUÊ

```
private function adjustResidualDifference(
    $expectedTotal,
    $calculatedPartialTotal,
    array $items,
    string $fieldName
): array
{
    $difference = bcsub((string)$expectedTotal, (string)$calculatedPartialTotal, 2);
    $lastIndex = array_key_last($items);

    if ($lastIndex === null || ! isset($items[$lastIndex][$fieldName])) {
        return $items;
    }

    if ($difference === '0.01') {
        $items[$lastIndex][$fieldName] = bcadd($items[$lastIndex][$fieldName], '0.01', 2);
    } elseif ($difference === '-0.01') {
        $items[$lastIndex][$fieldName] = bcsub($items[$lastIndex][$fieldName], '0.01', 2);
    }

    return $items;
}
```

POR QUE

? Por que exatamente '0.01'?

? Por que não '0.02'?

? E por que diabos o ajuste é feito SÓ no último item?



# Agora, o Código Explica o "Porquê"

Isto é ir "Além dos Comentários"

```
/**
 * Corrige diferenças residuais de 1 centavo em pagamentos parciais.
 *
 * Em cálculos contábeis, operações sucessivas com casas decimais podem
 * gerar diferenças de R$0,01 devido a arredondamentos.
 *
 * Essa função ajusta automaticamente o último item do lançamento
 * adicionando ou subtraindo 0,01 quando necessário, para garantir que o total final
 * coincida exatamente com o valor esperado.
 *
 * O objetivo **não é mascarar erros de cálculo**, mas apenas eliminar pequenas
 * diferenças técnicas inevitáveis em operações com precisão limitada.
 *
 * @param float|string $expectedTotal Valor total esperado.
 * @param float|string $calculatedPartialTotal Soma dos itens calculados (sem ajuste).
 * @param array<int, array<string, float|string>> $items Lista de itens com valores numéricos.
 * @param string $fieldName Nome do campo numérico que será ajustado no último item.
 * @return array Retorna o array de itens ajustado, se necessário.
 */
private function adjustResidualDifference(
    $expectedTotal,
    $calculatedPartialTotal,
    array $items,
    string $fieldName
): array
{
    $difference = bcsub((string)$expectedTotal, (string)$calculatedPartialTotal, 2);
    $lastIndex = array_key_last($items);

    if ($lastIndex === null || ! isset($items[$lastIndex][$fieldName])) {
        return $items;
    }

    if ($difference === '0.01') {
        $items[$lastIndex][$fieldName] = bcadd($items[$lastIndex][$fieldName], '0.01', 2);
    } elseif ($difference === '-0.01') {
        $items[$lastIndex][$fieldName] = bcsub($items[$lastIndex][$fieldName], '0.01', 2);
    }
}
```

? Por que '0.01'?



"...diferenças de R\$0,01 devido a arredondamentos em cálculos contábeis."

? Por que o último item?



"...para garantir que o total final coincida exatamente com o valor esperado."

? Qual o objetivo?



"Eliminar pequenas diferenças técnicas... não mascarar erros de cálculo."



# Os 3 Níveis de uma Documentação de Valor

Um framework para comunicar o "**Porquê**", o "**Onde**" e o "**Contrato**".

</\*\*>

Nível 1: O CÓDIGO

O "PORQUÊ"

---

**Público:** Quem está lendo a implementação.



Nível 2: A ARQUITETURA

O "ONDE"

---

**Público:** Quem precisa da visão geral.

}

Nível 3: A API

O "CONTRATO"

---

**Público:** Quem vai consumir o sistema.



# Nível 1: Documentando o "Porquê" com PHPDoc

Vá além da sintaxe, capture a intenção

O que a IDE faz por você (Ruído)

```
/**
 * @param float $valorDoPedido
 * @param string $estado
 * @return float
 */
function calcularFrete(float $valorDoPedido, string $estado): float
{
    $frete = self::VALOR_FIXO_FRETE;

    if ($valorDoPedido > self::VALOR_MINIMO_PEDIDO_ISENCAO_FRENTE
        && $estado === self::ESTADO_COM_FRETE_GRATIS
    ) {
        $frete = 0.00;
    }

    return $frete + self::TAXA_MANUSEIO;
}
```

? Qual a política de isenção de frete?

? De onde vêm essas constantes mágicas?

? A taxa de manuseio é somada no frete grátis?



# Nível 1: Documentando o "Porquê" com PHPDoc

Vá além da sintaxe, capture a intenção

O Foco Deve Ser o "Porquê":

```
/**
 * Calcula o valor final do frete, aplicando regras de isenção e adicionando a taxa de manuseio.
 *
 * A regra de isenção (frete = 0.00) é ativada se o pedido ultrapassar o valor
 * mínimo (self::VALOR_MINIMO_PEDIDO_ISENCAO_FRENTE) E for destinado ao
 * estado promocional (self::ESTADO_COM_FRETE_GRATIS).
 *
 * @note IMPORTANTE: A taxa de manuseio (self::TAXA_MANUSEIO) é SEMPRE
 * somada ao valor final, mesmo em casos de frete grátis.
 * Esta taxa não faz parte da política de isenção de frete.
 *
 * @param float $valorDoPedido O valor total dos produtos no carrinho.
 * @param string $estado A sigla do estado de destino (ex: 'SP').
 * @return float O valor final a ser cobrado (frete calculado + taxa de manuseio).
 *
 * @see self::VALOR_FIXO_FRETE
 * @see self::VALOR_MINIMO_PEDIDO_ISENCAO_FRENTE
 * @see self::ESTADO_COM_FRETE_GRATIS
 * @see self::TAXA_MANUSEIO
 */
function calcularFrete(float $valorDoPedido, string $estado): float
{
    $frete = self::VALOR_FIXO_FRETE;

    if ($valorDoPedido > self::VALOR_MINIMO_PEDIDO_ISENCAO_FRENTE
```

✓ A isenção é ativada acima do valor X para o estado Y.

✓ As tags @see apontam para as constantes.

✓ SIM! O @note avisa que a taxa é sempre somada



# Nível 2: Documentando o "Onde" (A Arquitetura)

Dando "zoom out" para a visão geral do sistema



## 1. O README.md

A "porta de entrada" do serviço

### Seções Essenciais:

- **O que é?** (Propósito do serviço)
- **Como executar?** (Instalação e run)
- **Arquitetura / Dependências** (Com quem ele fala?)
- **Variáveis de Ambiente**



## 2. O Diagrama (Simples)

O "mapa" do fluxo de dados.

### Exemplo:

**[Usuário] → [API Gateway] → [Serviço de Frete]**

**Responde:** "De onde vem a chamada e para onde ela vai?"



## 3. O ADR (Decisão)

O "porquê" da arquitetura.

### Estrutura Essencial:

**Título:** (Ex: "Uso de RabbitMQ")  
**Contexto:** (Qual era o problema?)  
**Decisão:** (O que foi escolhido)  
**Justificativa:** (Por que isso e não outro?)



# Nível 3: Documentando o "Contrato" (A API)

A documentação como "ponte" entre o Backend e o Consumidor.

## 1. O CÓDIGO (A Fonte da Verdade)

### Anotações (swagger-php) no Controller

```
/**
 * @OA\Info(
 *   version="1.0.0",
 *   title="Nome da Minha API",
 *   description="Descrição de como a API funciona",
 *   @OA\Contact(
 *     email="seu-email@dominio.com"
 *   )
 * )
 */
class FreteController extends Controller
```

```
/**
 * Você pode usar o DocBlock tradicional...
 * @OA\Get(
 *   path="/api/frete",
 *   summary="Calcula o valor final do frete",
 *   @OA\Parameter(
 *     name="valorDoPedido",
 *     in="query",
 *     required=true,
 *     @OA\Schema(type="number", format="float")
 *   ),
 *   @OA\Parameter(
 *     name="estado",
 *     in="query",
 *     required=true,
 *     @OA\Schema(type="string", example="SP")
 *   ),
 *   @OA\Response(
 *     response=200,
 *     description="Cálculo bem-sucedido",
 *     @OA\JsonContent(
 *       type="object",
 *       @OA\Property(property="valor_frete", type="number", format="float", example=15.50)
 *     )
 *   )
 * )
 */
public function calcularFreteApi(Request $request)
{
    // ...sua lógica da função calcularFrete() vai aqui...
    // return response()→json(['valor_frete' => $valorFinal]);
}
```



# Nível 3: Documentando o "Contrato" (A API)

A documentação como "ponte" entre o Backend e o Consumidor.

## 2. O CONTRATO (Interface do Consumidor)

The screenshot shows the Swagger UI interface for an API. At the top, there's a Swagger logo and a dropdown menu to 'Select a definition' with 'DOCS API' selected. The main title is 'Nome da Minha API' with version '1.0.0' and 'OAS 3.0' tags. Below the title, there's a URL 'http://127.0.0.1:8000/docs?api-docs.json', a description 'Descrição de como a API funciona', and a link 'Contact the developer'. A 'Filter by tag' input field is present. The 'default' tab is active, showing a 'GET' endpoint '/api/frete' with the description 'Calcula o valor final do frete'. Below this, there's a 'Parameters' section with two query parameters: 'valor do Pedido' (required, number) and 'estado' (required, string). A 'Try it out' button is in the top right of the parameters section. The 'Responses' section shows a 200 status code with the description 'Cálculo bem-sucedido' and a 'Media type' dropdown set to 'application/json'.

Swagger  
Supported by SMARTBEAR

Select a definition DOCS API

### Nome da Minha API 1.0.0 OAS 3.0

<http://127.0.0.1:8000/docs?api-docs.json>

Descrição de como a API funciona

[Contact the developer](#)

Filter by tag

#### default

**GET** **/api/frete** Calcula o valor final do frete

Você pode usar o DocBlock tradicional...

**Parameters** Try it out

Name	Description
<b>valor do Pedido</b> <span>★ required</span> number(float) (query)	<input type="text" value="valor do Pedido"/>
<b>estado</b> <span>★ required</span> string (query)	<input type="text" value="SP"/>

**Responses**

Code	Description	Links
200	Cálculo bem-sucedido	No links

Media type



# Por que esta abordagem funciona

---

- **Fonte Única da Verdade:** A documentação vive junto com o código. Se o código muda, a documentação muda.
- **Fim da Desatualização:** Adeus ao "esqueci de atualizar o Postman / Wiki".
- **Interface Interativa:** O consumidor (Frontend, outro dev) pode testar a API diretamente no navegador.



# Criando uma Cultura de Documentação

---

Não é sobre "fiscalizar", é sobre "colaborar"



## 1. Na Definição de "Pronto"

---

Inclua "documentar o porquê" como critério de aceite da task.  
Se a regra de negócio não estiver clara, a task não está pronta.



## 2. No Code Review

---

Use a documentação para guiar a revisão. A melhor pergunta não é "Faltou documentar", mas sim "Não entendi o porquê disso, podemos documentar?"



## 3. Comece Pequeno

---

Ninguém documenta um sistema legado inteiro em um dia. Escolha UMA parte crítica do sistema e melhore a documentação dela. Crie o hábito.



# Resumo das Ideias-Chave

---

**Nível 1: O CÓDIGO** → Explica o "PORQUÊ"

**Nível 2: A ARQUITETURA** → Explica o "ONDE"

**Nível 3: A API** → Define o "CONTRATO"

**Documente o "porquê", não o "o quê".**

***/\*\* "A documentação é um presente para o seu 'eu' do futuro." \*/***



# Obrigado!

## Perguntas?

---

@param \$person **João Vitor Santos**

 <https://www.linkedin.com/in/joao-vitorss/>