

Guia de Desenvolvimento Site/APP

Use Docker First - Then Learn About It Later!

Introdução

Este documento descreve um conjunto de normas, convenções e diretrizes para codificação.

Estes conceitos são baseados em princípios de engenharia de software que levam ao um entendimento intuitivo do código com ênfase em manter e melhorar novos projetos.

Além disso, seguindo estes padrões de codificação a produtividade aumentará consideravelmente.

A experiência mostra que, ao dispendermos de tempo para escrever código de alta qualidade desde o início, teremos mais rapidez para modificá-lo durante o processo de manutenção.

Finalmente, após um conjunto comum de padrões de codificação a equipe de desenvolvedores ficará muito mais produtiva.

Palavras Chaves

“MUST” (DEVE);

“MUST NOT” (NÃO DEVE);

“REQUIRED” (OBRIGATÓRIO);

“SHALL” (TEM QUE);

“SHALL NOT” (NÃO TEM QUE);

“SHOULD” (DEVERIA);

“SHOULD NOT” (NÃO DEVERIA);

“RECOMMENDED” (RECOMENDADO);

“MAY” (PODE);

“OPTIONAL” (OPCIONAL).

Padrões de código

Padrões de codificação são importantes porque levam a uma maior coerência dentro de seu próprio código e o código da equipe. Maior consistência leva o código a ser mais fácil de entender, o que significa que é mais fácil de desenvolver e de manter. Isso reduz o custo total das aplicações que criamos.

Codificação

A codificação padrão, deve ser UTF-8.

O código PHP deve usar as tags `<?php ?>` ou short-echo `<?= ?>`. Não utilize `<? ?>`.

NÃO DEVE existir um limite absoluto de caracteres em uma linha; O limite relativo de caracteres em uma linha DEVE ser de 120 caracteres; Verificadores de estilo de código automatizados DEVEM avisar caso uma linha ultrapasse 120 caracteres, mas NÃO DEVE acusar erro; Linhas NÃO DEVERIAM possuir mais de 80 caracteres; Linhas maiores que 80 caracteres DEVEM ser quebradas em múltiplas linhas de 80 caracteres ou menos.

Considere que as linhas devem ter no máximo 80 caracteres. Se em algum caso for necessário ultrapassar este limite, você tem até 120 caracteres. Não é recomendado ter linhas maiores que 120 caracteres, nos casos de linhas muito grandes, basta dividir em mais linhas de até 80 caracteres.

Linhas que não estão em branco NÃO DEVEM ter espaços após o conteúdo da mesma.

Linhas em branco PODEM ser adicionadas caso você ache que vá facilitar a interpretação do código.

NÃO DEVE haver mais que uma declaração por linha.

Códigos PHP DEVEM ser indentados com 4 espaços ao invés de TAB. (Ajuste seu Editor/IDE para isso)

As palavras-chave (e constantes) “true”, “false” e “null” DEVEM ser escritas com letras minúsculas.

Namespace and Class Names

Quando um namespace for definido, DEVE haver uma linha em branco após a definição deste.

Quando existente, todas as declarações “use” DEVEM estar após a declaração do namespace.

DEVE existir apenas um “use” por declaração.

As palavras-chave “extends” e “implements” DEVEM ser declaradas sempre na mesma linha do nome da classe.

Listas de “implements” PODEM ser divididas em múltiplas linhas, onde cada linha subsequente é indentada uma vez. Quando fazer isso, o primeiro item da lista DEVE estar na linha seguinte, e DEVE haver apenas uma interface por linha.

As classes devem ser declaradas utilizando o padrão chamado pelo PHP-FIG de StudlyCaps.

```
class MinhaClasse extends OutraClasse implements
    MinhaInterface,
    MinhaInterfaceDois
```

A visibilidade DEVE ser declarada em todas as propriedades.

A palavra-chave “var” NÃO DEVE ser utilizada para declarar uma propriedade.

NÃO DEVE haver mais de uma propriedade por declaração.

```
public $varOne;
public $varTwo;
public $varThree;
```

Métodos e Funções

A visibilidade DEVE ser declarada em todos os métodos.

Nomes de métodos NÃO DEVEM iniciar com underscore (_) para indicar visibilidade “protected” ou “private”.

NÃO DEVE existir um espaço em branco após o nome de um método; A chave de abertura do método DEVE ficar na próxima linha do nome do método, a chave de fechamento DEVE ficar logo após o corpo do método; NÃO DEVE haver um espaço em branco após o parêntese de abertura; NÃO DEVE existir um espaço em branco antes do parêntese de fechamento.

Na lista de argumentos dos métodos NÃO DEVE haver espaço antes de cada vírgula, mas DEVE haver um espaço após cada vírgula.

A lista de argumentos dos métodos PODE ser dividida em múltiplas linhas, onde cada linha é indentada uma vez. Quando fazer isso, o primeiro item da lista DEVE estar na linha seguinte, DEVE haver apenas um argumento por linha e o parêntese de fechamento da lista e a chave de abertura do corpo do método DEVEM estar na mesma linha, com apenas um espaço entre eles.

Na lista de argumentos, NÃO DEVE haver um espaço antes de cada vírgula e DEVE haver um espaço após cada vírgula.

Argumentos com valores padrão devem ir no final da lista.

Listas de argumentos podem ser divididas em várias linhas, onde cada linha subsequente é recuada uma vez. Ao fazer isso, o primeiro item da lista DEVE estar na próxima linha e DEVE haver apenas um argumento por linha.

Quando a lista de argumentos é dividida em várias linhas, o parêntese de fechamento e a chave de abertura DEVEM ser colocados juntos em sua própria linha, com um espaço entre eles.

Quando você tem uma declaração de tipo de retorno presente, DEVE haver um espaço após os dois pontos seguido pela declaração de tipo. Os dois pontos e a declaração DEVEM estar na mesma linha que a lista de argumentos entre parênteses, sem espaços entre os dois caracteres.

Quando presentes, as palavras-chave “abstract” e “final” DEVEM preceder a declaração de visibilidade do método ou propriedade.

Os nomes dos métodos DEVEM ser declarados em camelCase por exemplo `getMethod()`, `calcTotalIncome()`, `total()`

```
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // method body
    }

    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // method body
    }
}
```

NOTA: UM MÉTODO DEVE FAZER EXATAMENTE O QUE ELA SE PROPÕE SEM EXCEÇÕES, por exemplo em um método de cálculo onde preciso formatar o valor de retorno de um cálculo.

```
public function getTotal($numA, $numB)
{
    return $this->formatNumber($numA * $numB);
}

public function formatNumber($param)
{
    return 'R$' . number_format($param, 2, ',', '.');
}
```

Chamadas de método e função

Ao fazer uma chamada de função ou método, NÃO DEVE haver um espaço entre o nome do método ou da função e o parêntese de abertura, NÃO DEVE haver um espaço após o parêntese de abertura e NÃO DEVE haver espaço antes do parêntese de fechamento. Na lista de argumentos, NÃO DEVE haver um espaço antes de cada vírgula e DEVE haver um espaço após cada vírgula.

Listas de argumentos podem ser divididas em várias linhas, onde cada linha subsequente é edentada uma vez. Ao fazer isso, o primeiro item da lista DEVE estar na próxima linha e DEVE haver apenas um argumento por linha. Um único argumento dividido em várias linhas (como pode ser o caso de uma função ou matriz anônima) não constitui a divisão da própria lista de argumentos.

```
<?php

bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);

$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
)

somefunction($foo, $bar, [
    // ...
], $baz);

$app->get('/hello/{name}', function ($name) use ($app) {
    return 'Hello ' . $app->escape($name);
});
```

Propriedades e Constantes

A visibilidade DEVE ser declarada em todas as propriedades.

A visibilidade DEVE ser declarada em todas as constantes se o seu projeto versão mínima do PHP suportar visibilidades constantes

A palavra-chave `var` NÃO DEVE ser usada para declarar uma propriedade.

NÃO DEVE haver mais de uma propriedade declarada por declaração.

Os nomes de propriedades NÃO DEVEM ser prefixados com um underscore (`_`) para indicar visibilidade protegida ou privada. Ou seja, um prefixo de sublinhado explicitamente não tem significado.

DEVE haver um espaço entre a declaração de tipo e o nome da propriedade.

DEVE ser declarados na língua inglesa.

DEVE ser concisa e suficiente.

NÃO DEVE ultrapassar 30 caracteres

Propriedades DEVEM ser declarados em camelCase por exemplo `$variable`, `$myVariable`, `$mySingleVariable`

Constantes de classe DEVEM ser declaradas em maiúsculas separadas por sublinhado.

```
<?php
namespace Vendor\Model;

class Foo
{
```

```
public const VERSION = '1.0';
private const DATE_APPROVED = '2012-06-01';
public $variable;
private $myVariable;
}
```

Estruturas de controle

As regras gerais de estilo para estruturas de controle são as seguintes:

DEVE haver um espaço após a palavra-chave da estrutura de controle

NÃO DEVE haver um espaço após o parêntese de abertura

NÃO DEVE haver um espaço antes do parêntese de fechamento

DEVE existir um espaço entre o parêntese de fechamento e a chave de abertura

O corpo da estrutura DEVE ser recuado uma vez

O corpo DEVE estar na próxima linha após a chave de abertura

A chave de fechamento DEVE estar na próxima linha após o corpo

O corpo de cada estrutura DEVE ser delimitado por chaves.

Isso padroniza a aparência das estruturas e reduz a probabilidade de introdução de erros à medida que novas linhas são adicionadas ao corpo.

if, elseif, else

A palavra-chave `elseif` DEVE ser usada em vez de `else if`, para que todas as palavras-chave de controle pareçam palavras únicas.

Expressões entre parênteses PODEM ser divididas em várias linhas, onde cada linha subsequente é recuada pelo menos uma vez. Ao fazer isso, a primeira condição DEVE estar na próxima linha. O parêntese de fechamento e a chaves de abertura DEVEM ser colocados juntos em sua própria linha, com um espaço entre eles. Operadores booleanos entre condições DEVEM estar sempre no início ou no final da linha, não uma mistura de ambos.

```
<?php

if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}

if (
    $expr1
    && $expr2
) {
    // if body
} elseif (
    $expr3
    && $expr4
) {
    // elseif body
}
```

switch, case

Uma estrutura de `switch` é semelhante à seguinte. Observe o posicionamento de parênteses, espaços e chaves. A declaração de `case` DEVE ser recuada uma vez no `switch`, e a palavra-chave `break` (ou outras palavras-chave de término) DEVEM ser indentadas no mesmo nível que o corpo do caso. DEVE haver um comentário como `//no break` quando a falha é intencional `case` com corpo preenchido. Expressões entre parênteses PODEM ser divididas em várias linhas, onde cada linha subsequente é recuada pelo menos uma vez. Ao fazer isso, a primeira condição DEVE estar na próxima linha. O parêntese de fechamento e a chaves de abertura DEVEM ser colocados juntos em sua própria linha, com um espaço entre eles. Operadores booleanos entre condições DEVEM estar sempre no início ou no final da linha, não uma mistura de ambos.

```
<?php

switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}

switch (
    $expr1
    && $expr2
) {
    // structure body
}
```

while, do while

Expressões entre parênteses PODEM ser divididas em várias linhas, onde cada linha subsequente é recuada pelo menos uma vez. Ao fazer isso, a primeira condição DEVE estar na próxima linha. O parêntese de fechamento e a chaves de abertura DEVEM ser colocados juntos em sua própria linha, com um espaço entre eles. Operadores booleanos entre condições DEVEM estar sempre no início ou no final da linha, não uma mistura de ambos.

```
<?php

while ($expr) {
    // structure body
}

while (
    $expr1
    && $expr2
)
```

```

    ) {
        // structure body
    }

    do {
        // structure body;
    } while ($expr);

    do {
        // structure body;
    } while (
        $expr1
        && $expr2
    );

```

for

Expressões entre parênteses podem ser divididas em várias linhas, onde cada linha subsequente é recuada pelo menos uma vez. Ao fazer isso, a primeira expressão DEVE estar na próxima linha. O parêntese de fechamento e a chaves de abertura DEVEM ser colocados juntos em sua própria linha, com um espaço entre eles.

Uma declaração `for` se parece com o seguinte. Observe o posicionamento de parênteses, espaços e chaves.

```

<?php

for ($i = 0; $i < 10; $i++) {
    // for body
}

for (
    $i = 0;
    $i < 10;
    $i++
) {
    // for body
}

```

foreach

Uma declaração `foreach` é semelhante à seguinte. Observe o posicionamento de parênteses, espaços e chaves.

```

<?php

foreach ($iterable as $key => $value) {
    // foreach body
}

```

try, catch, finally

Um bloco `try-catch-finally` se parece com o seguinte. Observe o posicionamento de parênteses, espaços e chaves

```
<?php

try {
    // try body
} catch (FirstThrowableType $e) {
    // catch body
} catch (OtherThrowableType | AnotherThrowableType $e) {
    // catch body
} finally {
    // finally body
}
```

Operadores

As regras de estilo para os operadores são agrupadas por arity (o número de operandos que eles usam). Quando o espaço é permitido em torno de um operador, vários espaços podem ser usados para fins de legibilidade. Todos os operadores não descritos aqui são deixados indefinidos.

Operadores unários

Os operadores de incremento/decremento NÃO DEVEM ter nenhum espaço entre o operador e o operando.

```
$i++;
++$j;
```

Os operadores de conversão de tipo NÃO DEVEM ter nenhum espaço entre parênteses:

```
$intValue = (int) $input;
```

Operadores binários

Todos os operadores binários aritmética, comparação, atribuição, bit a bit, lógico, string e tipo DEVEM ser precedidos e seguidos por pelo menos um espaço:

```
if ($a === $b) {
    $foo = $bar ?? $a ?? $b;
} elseif ($a > $b) {
    $foo = $a + $b * $c;
}
```

Operadores ternários

O operador condicional, também conhecido como operador ternário, DEVE ser precedido e seguido por pelo menos um espaço em torno dos caracteres ? e :

```
$variable = $foo ? 'foo' : 'bar';
```

Quando o operador condicional é omitido, o operador DEVE seguir as mesmas regras de estilo que outros operadores de comparação binária:

```
$variable = $foo ?: 'bar';
```

Documentação de Código

Como padrão de desenvolvimento utiliza-se o PHPDoc para documentação de código, dessa forma padronizando e podendo ser reutilizado para gerar documentação posterior.

PHPDoc é uma parte da documentação que prove informações no aspecto da “estrutura dos elementos”.

“É importante notar que o PHPDoc e o DocBlock são duas entidades separadas. O DocBlock é a combinação de um DocComment, que é um tipo de comentário, e uma entidade PHPDoc é a que contém a sintaxe conforme descrito nesta especificação.”

Segue a coleção de elementos que DEVEM ser precedidos de DocBlock:

- require(_once)
- include(_once)
- class
- interface
- trait
- function (incluindo métodos)
- propriedades
- constantes
- variáveis escopo local e global.

O docBlock DEVE preceder o elemento estrutural

Tags padrão

Dentre as tags disponíveis no PHPDoc, estas seriam as mais relevantes para o projeto Revenda Mais.

	Tag	Elemento	Descrição
1	api	Métodos	declara que os elementos são adequados para consumo por terceiros.
2	copyright	Class	documenta as informações de direitos autorais do elemento associado. (Utilizar somente na classe)
3	depreca ted	qualquer	indica que o elemento associado está obsoleto e pode ser removido em uma versão futura.
4	ignore	qualquer	Evita a documentação de um elemento
5	internal	qualquer	Define descrição de elemento como privada, interna ao projeto e que não deve ser exibida na documentação final
6	link*	qualquer	Mostra um link dentro da documentação
7	param	Método, Function	Parâmetro de função ou método
8	return	Método, Function	Tipo de retorno de função ou método
9	see	qualquer	Link para a documentação de um elemento

10	source*	qualquer	Exibe o código-fonte de uma função ou método na descrição longa
11	todo	qualquer	Modificações a serem feitas
12	uses	qualquer	descrição de como o elemento é usado
13	var	Propriedades	Um atributo
14	inheritdoc*	Classe, Método	Usada para que as subclasses herdem a descrição detalhada de suas superclasses
15	package	Classe	Categoriza o elemento associado em um agrupamento ou subdivisão lógica.

“* Tags inline, ou seja podem ser utilizadas em qualquer lugar dentro do bloco da documentação”

```
/**
 * inline tags demonstration
 *
 * this function works heavily with {@link foo()} to rule the world.
If I want
 * to use the characters "{@link" in a docblock, I just use "{@}
link." If
 * I want the characters "{@*}" I use "{@}*}"
 */
```

CLASSE OU INTERFACES

Uma classe ou interface DEVE usar as tag's abaixo:

- @copyright

Funções e métodos

Uma função ou método DEVE usar as tag's abaixo:

- @param
- @return
- @api

Constantes e propriedades

Uma constante ou propriedade DEVE usar as tag's abaixo:

@var

```
/**
 * Esta classe funciona como um exemplo de onde posicionar um DocBlock.
 * @copyright 2020 - Revenda-Mais
 */
class Foo
{
    /** @var string|null $title contém o título para Foo */
    protected $title = null;

    /** @var int $int Este é um contador. */
    $int = 0 ;
}
```

```

// não deve haver docblock aqui
$int ++;

/**
 * Define o título de uma linha.
 *
 * @api
 * @param string $title Texto para o título.
 *
 * @return void
 */
public function setTitle($title)
{
    // Não deve haver docBlock aqui
    $this->title = $title;
}
}

```

Dúvidas sobre o uso das Tag's de PHPDoc pode-se consultar a seguinte documentação [PSR-PHPDoc](#).

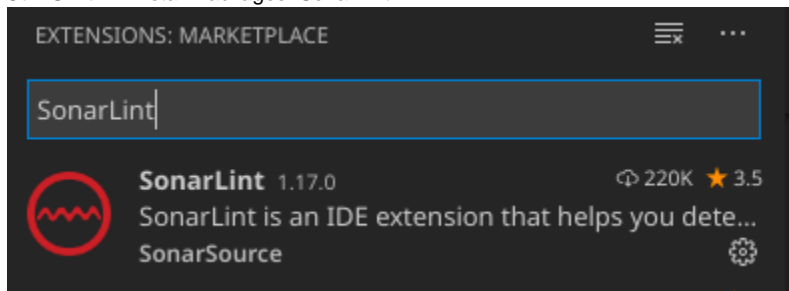
Configuração do SonarLint VSCode

É necessário proceder com a configuração do SonarLint conectado ao SonarQube para análise de qualidade de código.

Proceder conforme instruções abaixo:

Instalação do SonarLint

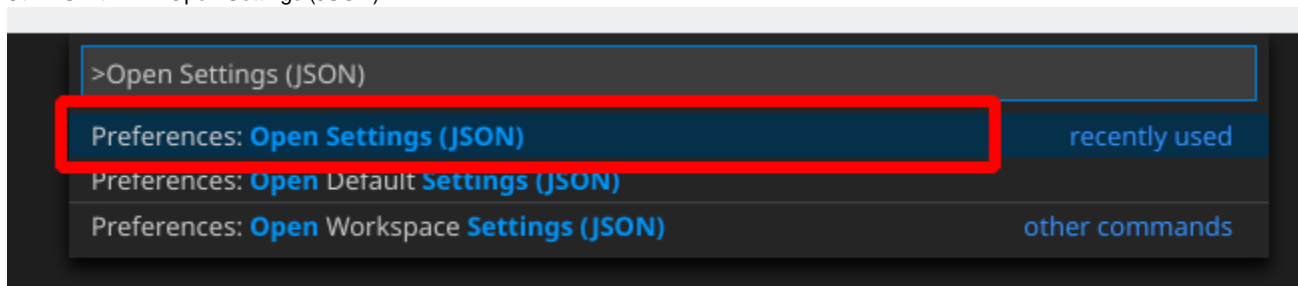
Ctrl+Shift+P Install Packages SonarLint



Proceder com a instalação normalmente.

Configuração SonarLint

Ctrl + Shift + P >Open Settings (JSON)



Colar o conteúdo abaixo nas suas configurações.

```
{
  "sonarlint.ls.javaHome": "/usr/lib/jvm/java-11-openjdk-amd64/",
  "sonarlint.rules": {},
  "sonarlint.connectedMode.connections.sonarqube": [
    {
      "connectionId": "Sonar",
      "serverUrl": "http://127.0.0.1:9011",
      "token": "4430744dcbcf3606aa10e7d1cbe9fbbf5a27028"
    }
  ],
  "sonarlint.output.showAnalyzerLogs": true,
  "sonarlint.output.showVerboseLogs": true,
}
```

Após a configuração do servidor proceder com a configuração do Workspace

Ctrl + Shift + P >Open Workspace Settings (JSON)

Colar o conteúdo abaixo nas suas configurações.

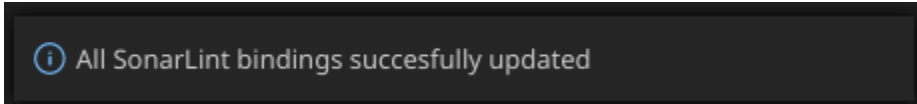
```
{
  "sonarlint.connectedMode.project": {
    "connectionId": "Sonar",
    "projectKey": "RevendaMais"
  }
}
```

Apos esse procedimento as Rules do projeto serão atualizadas e seu projeto será analisado conforme é desenvolvido.

Nota: A atualização é efetuada a cada abertura do VSCode, mas é importante forçar um update ao menos uma vez ao dia com o seguinte procedimento.

Ctrl + Shift + P >Update all project bindings to SonarQube/SonarCloud

Será exibida a mensagem de sucesso após a atualização.



Configuração efetuada.

FEITO ATÉ AQUI

Tamanhos máximos permitidos

- a) Arquivo: 500 linhas;
- b) Método: 30 linhas;
- c) 100 Caracteres por linha.

Regras do nível de criticidade Erro

Explicar aqui tudo relacionado a Erros.

PreserveStackTrace

Ao lançar uma exceção o programador deve ter o cuidado de preservar a pilha de exceções.

EqualsNull

CompareObjectsWithEquals

Entidades

Ao criar uma classe referente a uma entidade é preciso implementar os seguintes objetos:

- a) VO
- b) DAO
- c) DAOImpl
- d) Transform
- e) .queries.xml (Query somente da entidade)

Evitar uso de atributos transientes nas entidades, pois quem utiliza o serviço não saberá quando um atributo deverá ser carregado ou não. Procurar disponibilizar estas operações no serviço responsável pela entidade/funcionalidade.

Não utilizar o tipo Short na camada das entidades.

Definir o tipo dos atributos sempre como objetos e não como tipos primitivos.

Arquitetura

Explicar a forma de arquitetura, com diagramas, ou exemplos:

Fachada

A fachada serve bla bla bla...

Serviço

Estrutura

Negócio

Filtros

Domínios

Configuração

Validações

Consultas

Exceção

Responsabilidades dos Serviços

Persistência

Testes unitários

Referências

Roteiro de Desenvolvimento

- ☒ Verificação FK do banco
- ☒ Criação de migration de update de tabela
- ☒ Executar geração do código com InfyOm
- ☒ Ajustes na model
 - ☒ Ajustar a documentação do swagger para os novos nomes e remover campos que não serão utilizados (a coluna deleted_at nunca será retornada)
 - ☒ Ajustar a mutação na propriedade "\$columns"
 - ☒ Ajustar os tipos de campos em "\$searchCasts"
 - ☒ Adicionar as colunas de retorno na função "toArray()"
 - ☒ Remover todos os comentários
- ☒ Ajustes do repositório
 - ☒ Remover campos não utilizados na busca da propriedade "\$fieldSearchable"
 - ☒ Ajustar a função "findAll()" para o retorno padrão da consulta, (reve_cod, status, ativo, etc..).
- ☒ Ajustes da Factory
 - ☒ Ajustar o preenchimento da colunas de acordo com o uso
- ☒ Colocar a rota para dentro do middleware JWT
- ☒ Ajustar o arquivo "phpunit.xml" adicionando o model criado
- ☒ Ajustar a controller
 - ☒ Ajustar toda a documentação do swagger
 - ☒ Ajustar a a documentação do Search que é replica do GET
- ☒ Copiar as regras da Controllers
- ☒ Copiar as regras da Model
- ☒ Verificar a tela que utiliza essa model e ajustar os retornos

Informe os links das telas que foram verificadas.

/application/index.php/marcas/create
application/index.php/veiculos/create

application/index.php/tpveiculos/create
application/index.php/veiculos/create

Roteiro de Desenvolvimento

- ☒ Verificação FK do banco
- ☒ Criação de migration de update de tabela
- ☒ Executar geração do código com InfyOm
- ☒ Ajustes na model
 - ☒ Ajustar a documentação do swagger para os novos nomes e remover campos que não serão utilizados (a coluna deleted_at nunca será retornada)
 - ☒ Ajustar a mutação na propriedade "\$columns"
 - ☒ Ajustar os tipos de campos em "\$searchCasts"
 - ☒ Adicionar as colunas de retorno na função "toArray()"
 - ☒ Remover todos os comentários
- ☒ Ajustes do repositório
 - ☒ Remover campos não utilizados na busca da propriedade "\$fieldSearchable"
 - ☒ Ajustar a função "findAll()" para o retorno padrão da consulta, (reve_cod, status, ativo, etc..).
- ☒ Ajustes da Factory
 - ☒ Ajustar o preenchimento da colunas de acordo com o uso
- ☐ Colocar a rota para dentro do middleware JWT
- ☒ Ajustar o arquivo "phpunit.xml" adicionando o model criado
- ☒ Ajustar a controller
 - ☒ Ajustar toda a documentação do swagger
 - ☒ Ajustar a a documentação do Search que é replica do GET
- ☒ Copiar as regras da Controllers
- ☒ Copiar as regras da Model
- ☐ Verificar a tela que utiliza essa model e ajustar os retornos

Informe os links das telas que foram verificadas.

application/index.php/Modelos/
application/index.php/veiculos/create