Universidade Federal de Santa Catarina
Curso de Tecnologias da Informação e Comunicação
Desenvolvimento de Sistemas Web
Professor: Adriano de Oliveira

Desenvolvimento de Sistemas Web

Prof.: Adriano de Oliveira
email: adriano.inovar@gmail.com
Turma: 05652 - 3-2020-2 e 5-1830-2

Universidade Federal de Santa Catarina
Curso de Tecnologias da Informação e Comunicação
Desenvolvimento de Sistemas Web
Professor: Adriano de Oliveira

PSR - PHP Standards Recommendations

PHP-FIG

"A ideia deste grupo é que representantes de projetos possam conversar sobre pontos em comum entre seus projetos e encontrar formas de trabalharmos juntos. Nosso público alvo somos nós mesmos, mas sabemos que a comunidade PHP está participando. Qualquer pessoa que queira adotar o que estamos fazendo é bem-vinda, mas este não é o nosso foco."

PHP-FIG

- ✓ O FIG é um grupo composto por representantes de grandes projetos em PHP, tais como como o CakePHP, Doctrine, Symfony 2, Drupal e Zend Framework 2.
- ✓ Este grupo busca criar padrões que todos esses projetos possam seguir, definindo assim um "formato global" para projetos PHP.
- ✓ Esses padrões são chamados de PSR - PHP Standard Recommendation.

PHP-FIG

- ✓ PSR-1: Basic Coding Standard
- ✓ PSR-2: Coding Style Guide
- ✓ PSR-3: Logger Interface
- ✓ PSR-4: Autoloading Standard

PHP-FIG

- ✓ As PSR definem elementos usando algumas chaves para identificar sua aplicação no projeto. Essas chaves são:
  - "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).

PSR-1: Basic Coding Standard

- ✓ Arquivos PHP DEVEM usar somente as tags **`<?php`** e **`<?=`**
- ✓ Arquivos PHP DEVEM usar somente a codificação UTF-8 sem BOM (Byte Order Mark)

PSR-1: Basic Coding Standard

- ✓ Arquivos PHP DEVERIAM ou declarar símbolos (classes, funções, constantes, etc.) ou causar efeitos colaterais (gerar output, modificar configurações no php.ini, etc.). Mas NÃO DEVERIAM fazer ambos.

- ✓ Nomes de classes DEVEM ser declaradas em StudlyCaps

PSR-1: Basic Coding Standard

- ✓ Constantes DEVEM ser declaradas totalmente com letras maiúsculas e separadas por underscores.
- ✓ Nomes de métodos DEVEM ser declarados em camelCase.

PSR-2: Coding Style Guide

- ✓ Todos os arquivos PHP DEVEM utilizar o padrão Unix de terminação de linha.
- ✓ Todos os arquivos PHP DEVEM terminar com uma única linha em branco.
- ✓ A tag de fechamento (?>) DEVE ser omitida em arquivos de PHP puro.

PSR-2: Coding Style Guide

- ✓ NÃO DEVE existir um limite absoluto de caracteres em uma linha;
- ✓ O limite relativo de caracteres em uma linha DEVE ser de 120 caracteres;
- ✓ Linhas que não estão em branco NÃO DEVEM ter espaços após o conteúdo da mesma.

PSR-2: Coding Style Guide

- ✓ 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.

PSR-2: Coding Style Guide

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

PSR-2: Coding Style Guide

- ✓ 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.

PSR-2: Coding Style Guide

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

PSR-2: Coding Style Guide

- ✓ A visibilidade DEVE deve ser declarada em todos os métodos.
- ✓ Nomes de métodos NÃO DEVEM iniciar com underscore (_) para indicar visibilidade "protected" ou "private".

PSR-2: Coding Style Guide

- ✓ NÃO DEVE existir um espaço em branco após o nome de um método;
- ✓ A chave de abertura do método DEVE ficar na mesma 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.

PSR-2: Coding Style Guide

- ✓ 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.

PSR-2: Coding Style Guide

- ✓ Quando presentes, as palavras-chave “abstract” e “final” DEVEM preceder a declaração de visibilidade do método ou propriedade.
- ✓ Quando chamar um método ou função, NÃO DEVE haver espaços entre o nome do método/função e o parêntese de abertura, NÃO DEVE haver espaços após o parêntese de abertura e NÃO DEVE haver um espaço antes do parêntese de fechamento; Na lista de argumentos, NÃO DEVE existir espaços antes das vírgulas, mas DEVE existir um espaço após cada vírgula.

PSR-2: Coding Style Guide

- ✓ Ao chamar um método ou função, a lista de argumentos PODE ser dividida em múltiplas linhas. Quando feito isto, o primeiro argumento DEVE estar na linha seguinte, e DEVE haver apenas um argumento por linha.

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

PSR-2: Coding Style Guide

- ✓ NÃO DEVE haver um espaço após o parêntese de abertura (quando aplicável);
- ✓ NÃO DEVE haver um espaço antes do parêntese de fechamento (quando aplicável);
- ✓ DEVE haver um espaço entre o parêntese de fechamento e a chave de abertura (quando aplicável);

PSR-2: Coding Style Guide

- ✓ O corpo da estrutura de controle (quando aplicável) DEVE ser indentado uma vez;
- ✓ A chave de fechamento (quando aplicável) DEVE ser colocada uma linha após do corpo da estrutura de controle.
- ✓ A palavra-chave “elseif” DEVE ser usada no lugar de “else if”.

PSR-2: Coding Style Guide

- ✓ A palavra-chave “case” DEVE estar indentada um nível a frente do respectivo switch, e a palavra-chave “break” DEVE estar no mesmo nível que o corpo do respectivo “case”; DEVE haver um comentário do tipo “// no break” sempre que não houver uma palavra-chave “break”.
- ✓ Closures (funções anônimas) DEVEM ser declaradas com um espaço após a palavra-chave “function”, e um espaço antes e depois da palavra-chave “use”;

PSR-2: Coding Style Guide

- ✓ A chave de abertura da closure DEVE estar na mesma linha, e a chave de fechamento DEVE estar na linha após o corpo da função;

- ✓ NÃO DEVE existir um espaço após o parêntese de abertura da lista de argumentos e NÃO DEVE existir um espaço antes do parêntese de fechamento;

PSR-2: Coding Style Guide

- ✓ Na lista de argumentos NÃO DEVE haver um espaço antes de cada vírgula, mas DEVE haver um espaço após cada vírgula;
- ✓ Os argumentos que tenham um valor padrão DEVEM ser colocados no final da lista de argumentos;

PSR-2: Coding Style Guide

- ✓ A lista de argumentos da closure PODE ser dividida em múltiplas linhas, onde cada linha é indentada uma vez. Quando isto for feito, o primeiro item da lista DEVE estar na próxima linha, e DEVE haver apenas um argumento por linha; Quando a lista de argumentos for dividida em várias linhas, o parêntese de fechamento e a chave de abertura DEVEM estar na mesma linha, separados por um espaço.

PSR-3: Logger Interface

- ✓ Chamar o método log(), especificando um nível de log, DEVE ter o mesmo efeito que chamar um método específico.
- ✓ Chamar o método log() informando um nível de log não especificado DEVE lançar uma exceção do tipo Psr\Log\InvalidArgumentException; Usuários (desenvolvedores) NÃO DEVEM utilizar um nível de log personalizado sem ter certeza de que a implementação atual suporte este nível.

PSR-3: Logger Interface

- ✓ Todos o métodos de log DEVEM aceitar uma string ou um objeto com o método __toString() como mensagem de log; Métodos PODEM ter tratamentos especiais para objetos recebidos. Se não for o caso, métodos DEVEM transformar o objeto em string.

PSR-3: Logger Interface

- ✓ A mensagem de log PODE possuir placeholders que PODEM ser substituídos por elementos do argumento $context; placeholders DEVEM corresponder a uma chave dentro do argumento $context; placeholders DEVEM estar dentro de chaves ({}); NÃO DEVE haver nenhum espaço em branco entre as chaves e o placeholder; placeholders DEVERIAM ser compostos apenas de caracteres alfanuméricos (A-Za-z0-9), underscore (_) e ponto (.).

PSR-3: Logger Interface

- ✓ Métodos PODEM implementar estratégias para escapar valores de placeholders e tradução de logs para exibição. Desenvolvedores (usuários do seu sistema) NÃO DEVERIAM pré-escapar valores de placeholders, uma vez que eles não sabem em que contextos esses valores podem ser exibidos.

PSR-3: Logger Interface

- ✓ Todos os métodos do logger aceita um array como parâmetro de contexto. Neste array, qualquer informação adicional sobre o log pode ser incluída. Os métodos DEVEM tratar esse array da melhor forma possível. Os valores contidos no parâmetros de contexto NÃO DEVEM provocar qualquer tipo de erro ou disparo de exceção.

PSR-3: Logger Interface

- ✓ Se um objeto do tipo Exception for passado dentro do array de contexto, ele DEVE estar armazenado na chave “exception” do array. Salvar log de exceções é um bem comum e ajuda os desenvolvedores a fazer o rastreamento de pilha (stack trace) da exceção. Quando existir a chave “exception” no array de contexto, os métodos DEVEM verificar se realmente se trata de uma exceção antes de fazer uso deste valor, visto que essa chave, assim como todas dentro do parâmetro de contexto PODE conter qualquer tipo de dado.

PSR-4: Autoloading Standard

- ✓ O nome completo da classe DEVE possuir um namespace raiz.
- ✓ O nome completo da classe PODE possui um ou mais sub-namespaces.
- ✓ O nome completo da classe DEVE terminar com o nome de uma classe.
- ✓ Underscores (_) NÃO DEVEM possuir efeito especial em nenhuma parte do nome completo da classe.

PSR-4: Autoloading Standard

- ✓ Caracteres alfabéticos nos namespaces e classes PODEM ser definidas em qualquer combinação de letras maiúsculas e minúsculas.
- ✓ Todos os nomes de classes DEVEM ser referenciados de forma sensível a capitalização (case-sensitive).

PSR-4: Autoloading Standard

- ✓ Quando estiver carregando um arquivo correspondente a um nome completo de classe (\Namespace\Classe): uma série de um ou mais namepaces, não incluindo o namespace global (primeiro \), DEVE corresponder a pelo menos um diretório base; Cada sub-namespace definido depois do namespace global (primeiro \) corresponde a um subdiretório dentro do diretório base. O nome do subdiretório DEVE ser escrito da mesma forma que o nome do sub-namespace (respeitando maiúsculas e minúsculas); A classe chamada depois dos namespaces DEVE corresponder a um arquivo .php. O nome do arquivo DEVE ser escrito da mesma forma que o nome da classe (respeitando maiúsculas e minúsculas)

PSR-4: Autoloading Standard

- ✓ Implementações de autoloader NÃO DEVEM lançar exceções, NÃO DEVEM gerar erros de nenhum nível e NÃO DEVERIAM retornar nenhum valor.

FONTE: PHP-FIG

PHP-FIG — PHP Framework Interop Group - PHP-FIG

Blog do Dave (davidlima.com.br)