

**CAPSTONE PROJECT 1**

**ROOMY SYSTEM MANAGEMENT**

**CODE STANDARD DOCUMENT**

Version 1.0

Date: Sep 2nd, 2018

**MENTOR** **: MSc. Phan Van Son**

**PROJECT TEAM** : **Blue Team**

**TEAM’S MEMBER** : Huynh Quoc Nhat

Le Hoang Quoc

Ho Trung Anh

Huynh Vu Ha Lan

Huynh Thi Thanh Van

**INTERNATIONAL SCHOOL OF DUY TAN UNIVERSITY**

**PROJECT INFORMATION**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project acronym** | RMY | | | | |
| **Project Tittle** | Roomy | | | | |
| **Start Date** | Aug 15, 2018 | | **End Date** | | Dec 5, 2018 |
| **Lead Institution** | International School, Duy Tan University | | | | |
| **Project Mentor & contact details** | MSc. Son, Phan Van  Email: [sonpvs@duytan.edu.vn](mailto:sonpvs@duytan.edu.vn)  Tel: 01675373265 | | | | |
| **Scrum Master & contact details** | Huynh Vu Ha Lan  Email: [huynhvuhalan.97@gmail.com](mailto:huynhvuhalan.97@gmail.com)  Tel: 0989342173 | | | | |
| **Team members** | **Name** | **Email** | | **Tel** | |
| Le Hoang Quoc | [quocle28292@gmail.com](mailto:quocle28292@gmail.com) | | 01657708105 | |
| Huynh Quoc Nhat | [huynhquocnhat97@gmail.com](mailto:huynhquocnhat97@gmail.com) | | 0905740256 | |
| Ho Trung Anh | [hotrunganhht1912@gmail.com](mailto:hotrunganhht1912@gmail.com) | | 09055258534 | |
|  | Huynh Thi Thanh Van | [thanhvandb97@gmail.com](mailto:thanhvandb97@gmail.com) | | 01229444661 | |

**CODE STANDARD DOCUMENT**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DOCUMENT NAME** | | |
| **Document Title** | Code Standard Document | | |
| **Author(s)** | Blue Team | | |
| **Role** |  | | |
| **Date** | 7 – Sep – 2018 | **File name:** | CodingStandard\_ver1.0 |
| **URL** |  | | |
| **Access** | Project and CMU Program | | |

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Person(s)** | **Date** | **Description** |
| **1.0** | Blue Team | 7-Sep-2018 | Create Code Standard for project |

**Document Approvals:** The following signatures are required for approval of this document.

|  |  |  |
| --- | --- | --- |
| MSc. Vo Van Son  *Mentor* |  | Date: |
| Huynh Vu Ha Lan  *Scrum Master* |  | Date: |

1. **PHP Coding Standard**
2. **Overview**

* Files MUST use only <?php and <?= tags.
* Files MUST use only UTF-8 without BOM for PHP code.
* Files SHOULD either declare symbols (classes, functions, constants, etc.) or cause side-effects (e.g. generate output, change .ini settings, etc.) but SHOULD NOT do both.
* Namespaces and classes MUST follow an “autoloading” PSR: [[PSR-0](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md), [PSR-4](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-4-autoloader.md)].
* Class names MUST be declared in StudlyCaps and can be in Vietnamese or English, name capital letter of each word.
* Class constants MUST be declared in all upper case with underscore separators and can be in Vietnamese or English, name capital letter of each word.
* No more than 3 nested loops
* There must be a comment for each function
* Method names MUST be declared in camelCase.

1. **Files**
   1. ***PHP Tags***

PHP code MUST use the long <?php ?> tags or the short-echo <?= ?> tags; it MUST NOT use the other tag variations.

* 1. ***Character Encoding***

PHP code MUST use only UTF-8 without BOM.

* 1. ***Side Effects***

-A file SHOULD declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or it SHOULD execute logic with side effects, but SHOULD NOT do both.

-The phrase “side effects” means execution of logic not directly related to declaring classes, functions, constants, etc., merely from including the file.

-“Side effects” include but are not limited to: generating output, explicit use of require or include, connecting to external services, modifying ini settings, emitting errors or exceptions, modifying global or static variables, reading from or writing to a file, and so on.

-The following is an example of a file with both declarations and side effects; i.e, an example of what to avoid:

<?php

// side effect: change ini settings

ini\_set('error\_reporting', E\_ALL);

// side effect: loads a file

include "file.php";

// side effect: generates output

echo "<html>\n";

// declaration

function foo()

{

// function body

}

The following example is of a file that contains declarations without side effects; i.e., an example of what to emulate:

<?php

// declaration

function foo()

{

// function body

}

// conditional declaration is \*not\* a side effect

if (! function\_exists('bar')) {

function bar()

{

// function body

}

}

1. **Namespace and Class Name**

-Namespaces and classes MUST follow an “autoloading” PSR: [PSR-0, PSR-4].

-This means each class is in a file by itself, and is in a namespace of at least one level: a top-level vendor name.

-Class names MUST be declared in StudlyCaps.

-Code written for PHP 5.3 and after MUST use formal namespaces.

**For example:**

<?php

// PHP 5.3 and later:

namespace Vendor\Model;

class Foo

{

}

Code written for 5.2.x and before SHOULD use the pseudo-namespacing convention of Vendor\_ prefixes on class names.

<?php

// PHP 5.2.x and earlier:

class Vendor\_Model\_Foo

{

}

1. **Class Constants, Properties, and Methods**

The term “class” refers to all classes, interfaces, and traits.

* 1. ***Constants***

Class constants MUST be declared in all upper case with underscore separators. For example:

<?php

namespace Vendor\Model;

class Foo

{

const VERSION = '1.0';

const DATE\_APPROVED = '2012-06-01';

}

* 1. ***Properties***

This guide intentionally avoids any recommendation regarding the use of $StudlyCaps, $camelCase, or $under\_score property names.

Whatever naming convention is used SHOULD be applied consistently within a reasonable scope. That scope may be vendor-level, package-level, class-level, or method-level.

* 1. ***Methods***

-Method names MUST be declared in camelCase()

-This guide extends and expands on PSR-1, the basic coding standard.

-The intent of this guide is to reduce cognitive friction when scanning code from different authors. It does so by enumerating a shared set of rules and expectations about how to format PHP code.

-The style rules herein are derived from commonalities among the various member projects. When various authors collaborate across multiple projects, it helps to have one set of guidelines to be used among all those projects. Thus, the benefit of this guide is not in the rules themselves, but in the sharing of those rules.

-The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

1. **Overview**

* Code MUST follow a “coding style guide” PSR [PSR-1].
* Code MUST use 4 spaces for indenting, not tabs.
* There MUST NOT be a hard limit on line length; the soft limit MUST be 120 characters; lines SHOULD be 80 characters or less.
* There MUST be one blank line after the namespace declaration, and there MUST be one blank line after the block of use declarations.
* Opening braces for classes MUST go on the next line, and closing braces MUST go on the next line after the body.
* Opening braces for methods MUST go on the next line, and closing braces MUST go on the next line after the body.
* Visibility MUST be declared on all properties and methods; abstract and final MUST be declared before the visibility; static MUST be declared after the visibility.
* Control structure keywords MUST have one space after them; method and function calls MUST NOT.
* Opening braces for control structures MUST go on the same line, and closing braces MUST go on the next line after the body.
* Opening parentheses for control structures MUST NOT have a space after them, and closing parentheses for control structures MUST NOT have a space before.

1. **General**
   1. ***Basic Coding Standard***

Code MUST follow all rules outlined in PSR-1.

* 1. ***Files***

All PHP files MUST use the Unix LF (linefeed) line ending.

All PHP files MUST end with a single blank line.

The closing ?> tag MUST be omitted from files containing only PHP.

* 1. ***Lines***

-There MUST NOT be a hard limit on line length.

-The soft limit on line length MUST be 120 characters; automated style checkers MUST warn but MUST NOT error at the soft limit.

-Lines SHOULD NOT be longer than 80 characters; lines longer than that SHOULD be split into multiple subsequent lines of no more than 80 characters each.

-There MUST NOT be trailing whitespace at the end of non-blank lines.

-Blank lines MAY be added to improve readability and to indicate related blocks of code.

There MUST NOT be more than one statement per line.

***6.4. Indenting***

-Code MUST use an indent of 4 spaces, and MUST NOT use tabs for indenting.

N.b.: Using only spaces, and not mixing spaces with tabs, helps to avoid problems with diffs, patches, history, and annotations. The use of spaces also makes it easy to insert fine-grained sub-indentation for inter-line alignment.

***6.5. Keywords and True/False/Null***

PHP keywords MUST be in lower case.

The PHP constants true, false, and null MUST be in lower case.

**7. Namespace and Use Declarations**

-When present, there MUST be one blank line after the namespace declaration.

-When present, all use declarations MUST go after the namespace declaration.

-There MUST be one use keyword per declaration.

-There MUST be one blank line after the use block.

**For example:**

<?php

namespace Vendor\Package;

use FooClass;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

// ... additional PHP code ...

**8. Classes, Properties, and Methods**

The term “class” refers to all classes, interfaces, and traits.

***8.1. Extends and Implements***

-The extends and implements keywords MUST be declared on the same line as the class name.

-The opening brace for the class MUST go on its own line; the closing brace for the class MUST go on the next line after the body.

<?php

namespace Vendor\Package;

use FooClass;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable

{

// constants, properties, methods

}

-Lists of implements MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one interface per line.

<?php

namespace Vendor\Package;

use FooClass;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements

\ArrayAccess,

\Countable,

\Serializable

{

// constants, properties, methods

}

***8.2. Properties***

-Visibility MUST be declared on all properties.

-The var keyword MUST NOT be used to declare a property.

-There MUST NOT be more than one property declared per statement.

-Property names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.

-A property declaration looks like the following.

<?php

namespace Vendor\Package;

class ClassName

{

public $foo = null;

}

***8.3. Methods***

-Visibility MUST be declared on all methods.

-Method names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.

-Method names MUST NOT be declared with a space after the method name. The opening brace MUST go on its own line, and the closing brace MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

-A method declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

<?php

namespace Vendor\Package;

class ClassName

{

public function fooBarBaz($arg1, &$arg2, $arg3 = [])

{

// method body

}

}

***8.4. Method Arguments***

-In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

-Method arguments with default values MUST go at the end of the argument list.

<?php

namespace Vendor\Package;

class ClassName

{

public function foo($arg1, &$arg2, $arg3 = [])

{

// method body

}

}

-Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument per line.

-When the argument list is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them.

<?php

namespace Vendor\Package;

class ClassName

{

public function aVeryLongMethodName(

ClassTypeHint $arg1,

&$arg2,

array $arg3 = []

) {

// method body

}

*}*

***8.5. Abstract, final, and static***

-When present, the abstract and final declarations MUST precede the visibility declaration.

-When present, the static declaration MUST come after the visibility declaration.

<?php

namespace Vendor\Package;

abstract class ClassName

{

protected static $foo;

abstract protected function zim();

final public static function bar()

{

// method body

}

}

***8.6. Method and Function Calls***

-When making a method or function call, there MUST NOT be a space between the method or function name and the opening parenthesis, there MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis. In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

<?php

bar();

$foo->bar($arg1);

Foo::bar($arg2, $arg3);

Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument per line.

<?php

$foo->bar(

$longArgument,

$longerArgument,

$muchLongerArgument

);

**9. Control Structures**

The general style rules for control structures are as follows:

• There MUST be one space after the control structure keyword

• There MUST NOT be a space after the opening parenthesis

• There MUST NOT be a space before the closing parenthesis

• There MUST be one space between the closing parenthesis and the opening brace

• The structure body MUST be indented once

• The closing brace MUST be on the next line after the body

The body of each structure MUST be enclosed by braces. This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

[***9.1. if, elseif, else***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#51-if-elseif-else)

An if structure looks like the following. Note the placement of parentheses, spaces, and braces; and that else and elseif are on the same line as the closing brace from the earlier body.

<?php

if ($expr1) {

// if body

} elseif ($expr2) {

// elseif body

} else {

// else body;

}

The keyword elseif SHOULD be used instead of else if so that all control keywords look like single words.

[***9.2. switch, case***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#52-switch-case)

A switch structure looks like the following. Note the placement of parentheses, spaces, and braces. The case statement MUST be indented once from switch, and the break keyword (or other terminating keyword) MUST be indented at the same level as the case body. There MUST be a comment such as // no break when fall-through is intentional in a non-empty case body.

<?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;

}

[***9.3. while, do while***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#53-while-do-while)

A while statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

while ($expr) {

// structure body

}

Similarly, a do while statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

do {

// structure body;

} while ($expr);

[***9.4. for***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#54-for)

A for statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

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

// for body

}

[***9.5. foreach***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#55-foreach)

A foreach statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

foreach ($iterable as $key => $value) {

// foreach body

}

[***9.6. try, catch***](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#56-try-catch)

A try catch block looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

try {

// try body

} catch (FirstExceptionType $e) {

// catch body

} catch (OtherExceptionType $e) {

// catch body

}

**10.** [**Closures**](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#closures)

Closures MUST be declared with a space after the function keyword, and a space before and after the use keyword.

The opening brace MUST go on the same line, and the closing brace MUST go on the next line following the body.

There MUST NOT be a space after the opening parenthesis of the argument list or variable list, and there MUST NOT be a space before the closing parenthesis of the argument list or variable list.

In the argument list and variable list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Closure arguments with default values MUST go at the end of the argument list.

A closure declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

<?php

$closureWithArgs = function ($arg1, $arg2) {

// body

};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {

// body

};

Argument lists and variable lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument or variable per line.

When the ending list (whether of arguments or variables) is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them.

The following are examples of closures with and without argument lists and variable lists split across multiple lines.

<?php

$longArgs\_noVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) {

// body

};

$noArgs\_longVars = function () use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

$longArgs\_longVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

$longArgs\_shortVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) use ($var1) {

// body

};

$shortArgs\_longVars = function ($arg) use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

Note that the formatting rules also apply when the closure is used directly in a function or method call as an argument.

<?php

$foo->bar(

$arg1,

function ($arg2) use ($var1) {

// body

},

$arg3

);

**11.** [**Conclusion**](about:reader?url=http%3A%2F%2Fwww.php-fig.org%2Fpsr%2Fpsr-2%2F#conclusion)

There are many elements of style and practice intentionally omitted by this guide. These include but are not limited to:

* Declaration of global variables and global constants
* Declaration of functions
* Operators and assignment
* Inter-line alignment
* Comments and documentation blocks
* Class name prefixes and suffixes
* Best practices

Future recommendations MAY revise and extend this guide to address those or other elements of style and practice.