**Table of Contents – Developer Materials**

[1 Developer Materials 2](#_Toc531245739)

[1.1 To Be Architecture 2](#_Toc531245740)

[1.2 Coding Standards 3](#_Toc531245741)

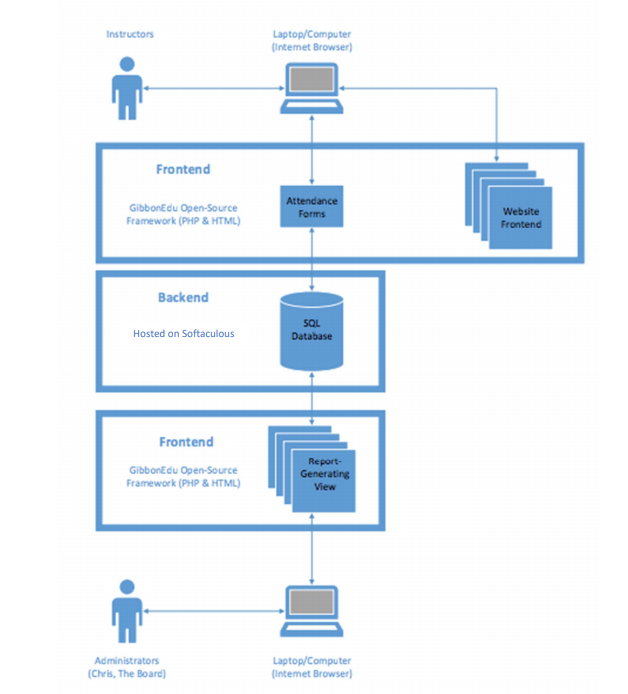
[1.3 Production Environment 4](#_Toc531245742)

[2 Installation Guide 5](#_Toc531245743)

[3 Operational Steps 5](#_Toc531245744)

# Developer Materials

## Updated To Be Architecture



## Coding Standards

During our development, we did not have to go into the original source code. However, if Chris would like to make future changes, he may need to make changes to Gibbon’s code. Listed below are the coding standards outlined by Gibbon that any future developer should adhere to:

## General

1. Avoid PHP Shortcodes - as of v8.0.00 Gibbon is moving away from <?in favour of <?php.

## Security

1. When using isActionAccessible(), always hard code the address, rather than relying on session, get, post or server variables.
2. For database connections, use the PDO class (instead of the more common mysql one), and parameterise any user-generated inputs. With PDO’s parameters, you do not need to escape your input.

## Code Style

Borrowed from Symfony: <http://symfony.com/doc/current/contributing/code/standards.html>

Gibbon follows the standards defined in the PSR-0, PSR-1, PSR-2 and PSR-4 documents.

Since a picture - or some code - is worth a thousand words, here’s a short example containing most features described below:

### Structure

* Add a single space after each comma delimiter;
* Add a single space around binary operators (==, &&, …), with the exception of the concatenation (.) operator;
* Place unary operators (!, –, …) adjacent to the affected variable;
* Always use identical comparison unless you need type juggling;
* Use Yoda conditions when checking a variable against an expression to avoid an accidental assignment inside the condition statement (this applies to ==, !=, ===, and !==);
* Add a comma after each array item in a multi-line array, even after the last one;
* Add a blank line before return statements, unless the return is alone inside a statement-group (like an if statement);
* Use return null; when a function explicitly returns null values and use return; when the function returns void values;
* Use braces to indicate control structure body regardless of the number of statements it contains;
* Define one class per file - this does not apply to private helper classes that are not intended to be instantiated from the outside and thus are not concerned by the PSR-0 and PSR-4 autoload standards;
* Declare the class inheritance and all the implemented interfaces on the same line as the class name;
* Declare class properties before methods;
* Declare public methods first, then protected ones and finally private ones. The exceptions to this rule are the class constructor and the setUp and tearDown methods of PHPUnit tests, which must always be the first methods to increase readability;
* Declare all the arguments on the same line as the method/function name, no matter how many arguments there are;
* Use parentheses when instantiating classes regardless of the number of arguments the constructor has;
* Exception and error message strings must be concatenated using sprintf;
* Calls to trigger\_error with type E\_USER\_DEPRECATED must be switched to opt-in via @ operator. Read more at Deprecations;
* Do not use else, elseif, break after if and case conditions which return or throw something;
* Do not use spaces around [ offset accessor and before ] offset accessor.

### Naming Conventions

* Use camelCase, not underscores, for variable, function and method names, arguments;
* Use underscores for option names and parameter names;
* Use namespaces for all classes;
* Prefix abstract classes with Abstract;
* Suffix interfaces with Interface;
* Suffix traits with Trait;
* Suffix exceptions with Exception;
* Use alphanumeric characters and underscores for file names;
* For type-hinting in PHPDocs and casting, use bool (instead of boolean or Boolean), int (instead of integer), float (instead of double or real);
* Don’t forget to look at the more verbose Conventions document for more subjective naming considerations.

### Service Naming Conventions

* A service name contains groups, separated by dots;
* The DI alias of the bundle is the first group (e.g. fos\_user);
* Use lowercase letters for service and parameter names;
* A group name uses the underscore notation.

### Documentation

* Add PHPDoc blocks for all classes, methods, and functions;
* Group annotations together so that annotations of the same type immediately follow each other, and annotations of a different type are separated by a single blank line;
* Omit the @return tag if the method does not return anything;
* The @package and @subpackage annotations are not used.
* Gibbon is released under the GNU General Public License, and the license block has to be present at the top of every PHP file, before the namespace.

**License**

* Gibbon is released under the GNU General Public License, and the license block has to be present at the top of every PHP file, before the namespace.

## Production Environment

The localhost version of Gibbon that can be installed free via Softaculous works as a production environment/sandbox. The developer can make changes to the source code, access the local MySQL database, and make changes to a version of the system that looks exactly like the deployed version. All the developer has to do is get the steps to set up the production environment, and then replicate that exactly in order to bring up the live system. An administrator account will be generated when the system is set up, and that account can be used to make all of the other system users.

# Installation Guide

**Go Live:**

1. Created subdomain for domain currently owned through
2. Set up MySQL database on DreamHost for use by Gibbon
3. Created SSL Certificate from letsencrypt.org to allow HTTPS
4. Download Gibbon manual installation file from gibbonedu.org
5. SCP (secure copy) the file to server
6. Unpack the file into the directory tied to the subdomain
7. Navigate to the subdomain (gibbon.cycleacademy.org) in a browser
8. Follow the instruction to set up an admin account
9. Connect the MySQL database using the on-screen prompts

NOTE: The above steps have been performed by Chris, so he has familiarity with the hosting platform and setup.

1. Using the Import Users feature, import the users from a CSV file
2. Using the Import Enrollment,

NOTE:We have imported 2017 users and classes, and have given Chris the file for future use. Also, instructions for importing can be found in the user materials we have provided or in Gibbon’s official documentation.

**Add Badges Module:**

1. Download Badges Module from: <https://gibbonedu.org/extend/>
2. Unzip and move Badges folder to /Applications/AMPPS/www/ghisallotest.com/gibbon/modules/
3. Select System Admin from Admin in the navbar
4. Select Manage Modules from the right sidebar
5. Click + to install the module

# Operational Steps

**Check System Status (deployed or production):**

1. Select System Admin under Admin in the navbar
2. Select System Check from the right sidebar
3. You can now view the status of the stack configuration

NOTE: Issues should only arise if there has been an update to Gibbon or one of the components of the LAMP stack. If this is the case, simply update the system through the UI as described below.

**Update System (deployed or production):**

1. Select System Admin under Admin in the navbar
2. Select Update from the right sidebar

NOTE: This step should only be necessary if the System Status page says an update is required (detailed above).

**Importing Old Data:**

1. Historical data can be imported using the Import Users feature (steps are outlined in the administrator materials).
2. The import will only import the users, their attendance data will not transfer over to the new system.

**Backing Up the Installation:**

* Gibbon suggests using a shell script to copy all files and database data to an external drive. Here is a sample of the backup script, you just need to fill in the different path variables:

#!/bin/bash

#Backup files in the Gibbon installation to the backup drive

sudo rsync -rlptDzvv -delete [gibbonInstallPath] [backupPath]

#Dump the Gibbon database and copy to the backup drive

sudo mysqldump -u [databaseUsername] -p[databasePassword] -h [databaseServer] [databaseName] > ~/[databaseName].sql

sudo rm [backupPath]/mysql/[databaseName].sql

sudo mv ~/[databaseName].sql /[backupPath]/[databaseName]/

#Backup the home directory, which is where this backup script may be stored

sudo rsync -rlptDzvv -delete [homeDirectoryPath] [backupPath]

#Backup /etc, which contains a lot of important server config files

sudo rsync -rlptDzvv -delete /etc [backupPath]

* It is suggested to use cron (a scheduler on Linux) to run the script periodically.

**User Administration (Included in the Administrator QRG and User Manual)**