I deleted the page where we had the company introduction for privacy purposes

# Codeception

## Abstract

We utilize the Codeception framework for our automated testing. Codeception is developed by a wide community with the contributors including top contributors from other PHP frameworks, including Phalcon and Laravel. Codeception was developed to be very flexible. This flexibility allows you to write your tests the same, regardless of the underlying framework you may use.

## PHPUnit

Codeception is built on top of PHPUnit. All Unit Tests all have complete access to everything PHPUnit gives you.

## Resources

All the information in this file can be found at the Codeception website at [http://codeception.com](http://codeception.com/). They have comprehensive documentation on how to build your suites and tests.

## PHPStorm

PHPStorm has built-in Codeception support. This will allow you to easily run your testing directly through PHPStorm. This does not however stop you from running in the command line.

## Suites

Tests are divided into suites. These suites can all be individually configured and allow you to separate core testing. Codeception is typically set up with three default suites: Unit, Functional, and Acceptance. Each suite is meant to serve a slightly different purpose.

The unit suite is no different from your normal unit-testing. Each test in the suite should test that a single class works as expected. You may liken this to the wall of a house.

The functional testing can be likened to the room of a house. Functional testing is used primarily in MVC testing and tests the functionality of the controller without running the code through a web browser.

Acceptance testing can be likened to the whole house. Acceptance testing runs through an actual browser using the Selenium software. It able to simulate a person clicking links, filling out fields, and/or submitting forms.

Any suite may be ran by running ‘./vendor/bin/codecept’ run [suite name]. You may run all suites by not specifying a suite to the command when it runs.

## Configuration

The primary configuration for Codeception is codeception.dist.yaml in the root of the project. You may override any configurations in this file locally with codeception.yaml. Each suite has its own configuration file in the tests directory that has a similar format. [suite name].suite.dist.yaml is for the repository config and [suite name].suite.yaml is used for local changes. For example, if you wanted to not populate the database in your acceptance testing, you may put the following into the file acceptance.suite.yaml:

modules:

config:

- Db:

populate: false

## Environments

One advanced feature that Codeception has is the ability to define ‘environments’ You may place additional configuration files in the ‘env’ folder of tests. These files can be passed to the command via the –env flag and allow you to apply any number of combinations of config settings. For example, if you wanted to test in both chrome and firefox, you could have one browser be the default and then create an environment file that changes it to firefox.

## ‘Test’ vs ‘Cest’

When developing Unit Tests, your file will be named ‘Test’ and extend ‘\Codeception\Util\Test’.

For all other suites, you can use the ‘Cest’ format instead. This format is setup to allow a bit easier

representation and configuration in your tests. All public methods of a Cest file are called for the testing.

## Code Coverage

Codeception supports code coverage. These files are all generated after the tests run and are place in the tests/\_output folder. You may activate code coverage using the ‘--coverage-html' flag to your command.

## Unit Testing

Because Codeception runs on top of PHPUnit, you may use all available functions in your unit

tests:

* $this->assertTrue(true)
* $this->assertSame($item, $item->method())
* $this->getMockBuilder(ClassName::class)->getMock().

In addition to this, the $this->tester variable gives you access to all methods that the various Codeception models activate. Visit the website to learn more.

* $this->tester->haveInDatabase(‘table’, [‘field’ => ‘value’])
* $this->tester->seeInDatabase(‘table’, [‘field’ => ‘value’])

## Functional Testing

Functional Testing works by simulating a browser request and running the controller. Every function will have the parameter $I. This parameter is the same as the tester object in unit testing.

## Acceptance Testing

Acceptance testing works by running the code through an actual browser, allowing you to also test javascript. The syntax is the same as functional testing. The modules for acceptance testing gives you methods like:

* $I->click()
* $I->seeElement()
* $I->fillField()

## Common Commands

* ./vendor/bin/codecept run – Run All Commands
* ./vendor/bin/codecept run unit – Run just the unit suite.
* ./vendor/bin/codecept run acceptance –env firefox - Run the acceptance suite with the added firefox environment
* ./vendor/bin/codecept generate:test unit ClassName – Generate a pre-written test called ClassName in the unit suite.

## Common Test Helper Actions

The specific variable used in the test will depend on the test type. In a unit test, you will use

‘$this->tester->’. In other suites (Cest type tests), you will use the parameter, typically called ‘$I’. For the

sake of example, only $I will be used here.

* $I->haveRecordInDatabase(‘table’, [‘field’ => ‘value’, ‘field2’ => ‘value2’]) - Add a record to the database.
* $I->seeInDatabase(‘table’, [‘field’ => ‘value’, ‘field2’ => ‘value2’]) - Returns true if the record is found in the database, false otherwise.
* $I->haveInSession(‘key’, ‘value’) - Adds a index/value to the session.
* $I->seeInSession(‘key’, ‘value’) - returns true fi the record is found in the session.
* $I->grabServiceFromContainer(‘service’) - retrieves a service from the dependency injection service.
* $I->addServiceToContainer(‘service’, ‘value’) - add service to the dependency injection service.
* $I->grabRecord(ModelName::class, [‘field’ => ‘value’]) - returns the model for the record that matches the criteria.
* $I->click([selector]) - Click an element.
* $I->seeElement([selector]) - Assert an element is there
* $I->waitForElement([selector]) - Wait For Element to Render
* $I->fillField([selector], ‘value’) - Fill form field
* $I->amOnPage(‘/some-url’) - Navigate to page;
* See the Documentation on the module to get all available methods.

## Common Assertions (Unit/Functional/Acceptance)

The specific variable used in the test will depend on the test type. In a unit test, you will use

‘$this->’. In other suites (Cest type tests), you will use the parameter, typically called ‘$I’. For the sake of example, only $this will be used here. It should be noted that the ‘Asserts’ module has to be enabled for functional and acceptance suites, but these work out of the box in unit tests.

* $this->assertEquals(‘some-value’, $this->testSubject->getValue());
* $this->assertSame($this->testSubject, $this->testSubject->doSomething());
* $this->assertTrue($this->testSubject->isTrue());
* $this->assertFalse($this->testSubject->isFalse());

## Selectors

When working on your tests, several methods require a selector (Indicated by [selector] above).

This can be any type of valid Dom Selector.

* #element
* .element-class
* //div/span/a[href=””]
* //\*/a[alt=”Some Alt”]

## Configuring PhpStorm

Before Codeception can be used with the built in PhpStorm support, you must first do a one- time configuration. Ensure that you have ran ‘composer install’ on the project first. Then, open the project settings dialog (preferences on OSX) and go to Languages and Frameworks -> PHP -> Testing Frameworks. Click the ‘+’ sign to add a new framework. Choose the menu option ‘Codeception Local’. In the ‘Path to Codeception executable’, click the browse button and navigate to the [Project Root]/vendor/bin/codecept. Once selected, you can click okay to get out of the settings dialog.

After you have setup codeception, you may add a new Run action. Use the ‘Edit Configurations’ option to open the config dialog and then click the ‘+’ and choose ‘Codeception’. The dialog will give you simple options to select the suite/options to run the appropriate test you wish to run. Alternatively, you may open any test file. In the file you will see a double arror icon (>>) in the left bar next to the line numbers. Clicking this will create a temporary run action to run that specific test.

## Example Test – Unit

class Something {

public function doSomething() {

//...

mysqli\_query($link, ‘Insert into table (a, b, c) values (1, 2, 3’);

return $this;

}

}

class SomethingTest extends Codeception\Util\Test { private $testSubject;

public function setUp(){

$this->testSubject = new Something();

}

public function testDoSomething() {

$this->assertSame($this->testSubject, $this->testSubject->doSomething());

$this->tester->seeInDatabase(‘table’, [‘a’ => 1, ‘b’ => 2, ‘c’ => 3]);

}

}

## Example Test - Functional/Acceptance

class SomethingCest {

public function testDoSomething(AcceptanceTester $I) {

$I->amGoingToTest(‘It Does Something’);

$I->amOnPage(‘/some-url’);

$I->fillField(‘#name’, ‘value’);

$I->fillField(‘.my-value’, ‘other-value’);

$I->click(‘#submit’);

$I->seeInDatabase(‘table’, [‘name’ => 'value’, ‘value’ => 'other-value']);

}

}

## Common Errors

* Cannot set Headers, Output Already Sent in Printer.php - This usually only occurs when running Codeception through PhpStorm. If you run through the command line, it should run fine. You can get around this by adding ‘if (header\_sent()) { $this-markTestSkipped(‘Headers Already

Sent’); }’ in the \_before action of your test. Understand though that this is a potential fail point.

Database Standards

## Overview

This document reviews the various company standards as they relate to the database. These include both best practices as well as general rules as determined by the CEO. It must be noted that many of these standards were implemented in the past few years. You will find that many legacy parts of the system do not follow these standards as a result.

## Updating Legacy Code/Standards

Due to the fragile nature of the application, there are no current plans to update the legacy code to match these standards.

## Global Tables

A table is considered global if all records are universally used for every account, regardless of whom is logged in. Most of the tables are not global. Such an example table is the `lob` and `system` tables.

## Null Columns

When working with existing tables, you must always allow null values. This is true even if you believe the column should never not have a value. Many of the legacy parts of the application may use direct SQL and will break if you do not allow null values. For new tables, `not null` is allowed.

## Default Values

You should specify default values for all columns that allow a `null` values. This is not necessary if the default value is null. In the case of date or datetime fields, leave the default as null.

## Naming Conventions

All tables and columns are to be in alternative camel case (AKA medial capitals) format. The first letter should be lower case, with all distinguishable words capitalized. When working with acronyms (IE: URL), treat the acronym as a word.

* columnName
* masterId
* name
* url
* pictureUrl

## Required Columns

All columns below should be added to every table created. This excludes tables deemed global.

* id – Unsigned int(11) Primary Key Unique Not Null - Record identifier
* masterId – Unsigned int(11) Foreign Key To `Config` - Account ID
* `tranDate` datetime NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP – Date Updated
* `insertDate` datetime NULL DEFAULT CURRENT\_TIMESTAMP - Date Inserted

## Common Columns

The columns below are some you will see occasionally throughout the system.

* active – TINYINT(1) NOT NULL DEFAULT 1 – Is the records considered ‘active’ and accessible. Often referred to as ‘Hidden’
* deleted – DATETIME NULL DEFAULT NULL – Soft Delete Column

## Foreign Keys

If a column references another table, the field name should be in the format of `tableField`. IE: The field containing the id of an associated office should be `officeId`. If you have more than one column referencing the same table, use the format of `typeTableField`. IE: agentEmployeeId. All such columns should be registered as a foreign key to the parent table using type `NO ACTION`. The foreign key name should follow the camelCase concention and be prefixed ‘fk’. IE: fkOfficeIdEmployeeId

## Stored Procedures, Triggers, and Views

We do not ever use stored procedures, triggers, or views.

## Common Legacy Column Names

These are some of the legacy column names you may see in the system and their purpose.

* Customer ID
  + custid
  + customer
  + customerID
  + customer\_id
* Office ID
  + office
  + officeid
  + officeID
* Employee ID
  + user
  + userID
  + employee
* Account ID
  + masterID
  + masterid
  + master\_id

## Prohibited Column Names

The following column names are prohibited from use as column name.

* Anything representing a query command. AKA any column that must be wrapped in backticks to work. IE: `drop` or `select` or `to`
* `source` - This plays havoc with the Phalcon ORM.

## Common Gotchas

Phalcon Default Values

Phalcon Models should have the default values set on properties. If not set, the ORM will explicitedly pass null through the queries.

# Employee Handbook

- Developer -

## Subject to Change

This document is subject to change at the discretion of senior management. Should such changes be made, you will be made aware.

## Additional Documents

In addition to this document, you should have also received several additional documents. Some of these documents must be signed and returned immediately to the VP of Application Development and will become part of your permanent profile. If you were not given any extra documents, be sure to request them. One of these documents will include the names of current developers and people of importance mentioned throughout this document.

## Infrastructure vs Code

Development is divided into two sub-departments: Infrastructure and Code. Infrastructure includes (but is not limited to) the database, frontend and backend servers. This document applies to the Code sub-department. The Code sub-department is maintained/managed by the VP of Application Development. The Infrastructure sub-department is maintained/managed by the President. Decisions regarding the code and development there in will be made by the VP of Application Development.

Decisions regarding the infrastructure and production environments will be made by the President. The departments will work together as necessary to ensure the efficiency of the system.

## Company Policies

All company policies apply. This includes (but isn’t limited to) dress and cleanliness. Such policies

are monitored and managed by the Human Resources (HR) Department.

## Human Resources (HR)

The HR department is available to employees at any time. Should you have a problem with a co- worker, or have problems with policies or the like, you may report it to the HR Representatives. This includes (but is not limited to): verbal abuse, disrespect, concerns with policies, concerns with co- workers. It is encouraged that you feel comfortable going to HR for any reason. They are there to assist you and will do so in a discreet manner. If, however, for any reason you feel uncomfortable going to HR, you may also go to the VP of Application Development. He or She will decide if it is appropriate to elevate to HR on your behalf.

## Disciplinary Action(s)/Termination

Throughout this document you will see the phrase ‘may result in disciplinary action’. Such actions may include anything from a simple warning or write up to a pay cut or immediate termination. All actions, excluding verbal warnings will go in your profile. Should enough such actions be added to your profile, it will result in a review to determine if termination is necessary.

## Annual Reviews

Work/Lunch Hours

The office hours are 9:00 AM till 6:00 PM with one hour for lunch. In certain circumstances, developers may be allowed to shift their work hours so long as they still get 8 hours in during the day. Such circumstances must be approved by the VP of Application Development. There is one exception to this: All developers are required to be at the morning meeting as scheduled (See Morning Meetings).

To ensure that there is always someone available in the event the system breaks, goes down, or an emergency comes up, lunch is taken in shifts starting at 11:30 AM till 1:30 PM. You will be assigned your specific lunch shift as a guideline. We understand that co-workers will often desire to eat with other co-workers. While this is encouraged, if going with them would result in there being no developers in the office, at least one person must stay behind, or the party must wait until other developers return.

Developers are given one hour for lunch. Should your lunch last longer than an hour, you are required to stay longer to make up for the extra time. If this happens repeatedly, it may result in disciplinary action.

In addition to the normal 9/6 work hours, developers are required to log a minimum of 32 hours of work towards individual tickets (See Tickets, Logging Hours). Failure to meet these hours may result in disciplinary action.

## Morning Meetings

There will be one meeting scheduled every Monday morning at 10:00 AM. These meetings will serve multiple purposes. This includes (but is not limited to): Discussions of Upcoming Projects, Distributing Tickets, Simple Training, and any other topics the VP of Application Development or management deem necessary. These meetings are mandatory for all developers, regardless of the developer shifted hours. Failure to show may result in disciplinary action. This meeting is not restricted to developers and may include people from other departments.

## On Call

All developers are considered ‘On-Call’ for emergency purposes. To this end, you must give the VP of Application Development a valid after number you may be reached at in the event you are needed to either answer questions regarding code or potentially work on said code.

In most instances, it will only be the VP of Application Development that contacts you. If at any point are receiving calls from anyone other than the VP of Application Development, be sure to report it.

## Working Weekends/Extra Hours

Developers are not required to work weekends or extra hours. However, there may be special circumstances that require them to do so (less than 10%). Such instances are for emergency repairs, failure to meet deadlines, critical deadlines, or for deploying updates that require the software to be in

limited use. You may be allowed to work remotely in certain circumstances. The VP of Application Development will make the appropriate decision and inform you if you will need work extra hours.

If you should feel at any time that you are working more hours than required, and such hours are not due to missed deadlines, please inform the VP of Application Development.

## Working Remotely

Developers are typically not allowed to work remotely. Should the need to work remotely arise, you must get permission from the VP of Application Development. Such instances include emergency work or the need to meet deadlines. If the VP of Application Development determines that it is necessary, he or she will get appropriate authorization from upper management.

## In-House Continuing Education/Training

The VP of Application Development will occasionally schedule an in-house continuing education/training session. These meetings will typically be scheduled to start at Noon and lunch will be provided by the company. All developers will be expected to attend unless told specifically by the VP of Application Development that their attendance is not necessary. Please note that the lunch is for those attending the training and is not available to those not required to attend.

## Release/Development Cycle

We release the new version of our software once per quarter. This equates to four major releases per year. Each cycle is three months long. The cycle is divided into two phases: The Release Cycle and the Development Cycle. The release cycle is offset by one month past our development cycle.

At the end of the development cycle, the Beta version of the software is put under lockdown. At this point the new Alpha version of the software will be spun off. All new development will go in the Alpha version, only bug fixes will go into the Beta version, and only mission critical bug fixes will go into the Stable release.

Over the next month, the Beta version is considered a release candidate. At this point, all non- developers should be testing the system as a final pre-release test to ensure the stability of the code and report any additional bugs.

After one month, the release cycle will activate. At this point, the Beta version will be rolled out to the Stable environment and the Alpha environment will be made the new Beta Environment.

## Break Fix Team

During each development cycle, one or more developers will be assigned to the break fix team. Excluding emergencies, the Break Fix team members will only be working on bug tickets or Rollbar issues (See Rollbar). At the end of every development cycle, the break fix team may rotate members to give those on the team the opportunity to work on specific projects or to give other developers the opportunity to ‘relax’ from normal projects.

The Break Fix team is expected to spend no more than one hour on any individual ticket/issue. If they are unable to figure out the ticket/issue in that time, they may inform the Traffic Organizer to get it reassigned to another developer.

## Agile Development

Excluding the developers on the Break Fix team, we utilize the agile development flow process. While there will be senior staff assigned to specific sections of the system, all staff will require a basic understanding and ability to work on any part of the system at any time.

Developers may be grouped together for specific projects, but the work will be divided among the developers of that group to ensure a quick completion of the project.

## Technologies

We utilize the following technologies in the software or for development. While knowledge of each technology is not required, it is recommended that all developers at least strive to obtain an entry level basic understanding of each.

* PHP 7.0
  + Phalcon 3.3.2
  + Composer
    - Laravel 5.5
    - Codeception
    - PHP-CS-Fixer
    - PHPStan
    - Infection
    - Other third-party packages
* Linux/Unix
* MariaDB 10.3
* Javascript
  + AngularJS 1.2
  + jQuery 1.12
  + VueJS
* HTML5
* CSS3
* Git Version Control Software (VCS)
  + GitLab
  + GitLab Runner (See Pipelines)
* Legacy Technologies
  + Dojo
  + Doctrine
  + Zend

## Tickets

All work must be tied to a ticket. Each week, you will be assigned specific tickets to work on. Each developer will be assigned a minimum of 32 estimated hours per week. If at any time you do not have any remaining tickets assigned to you for the week, talk with the Traffic Organizer. He or She will shift any items from your next weeks’ queue to this week or pull tickets from the backlog.

At no point should you be working on anything without an associated ticket. If you receive a task to do a job via an alternative source (IE: email, word of mouth, etc), you are required to immediately report the task to the Traffic Organizer. The Traffic Organizer will then determine if you are to continue working on the task, if it is better suited for someone else, or if your schedule permits you to work on said task. Under no circumstances should you ever be working on any task without the Traffic Organizer knowing. Failure to complete your assigned 32 hours of work may result in having to work late to meet deadlines or keep on schedule. This is especially true if you were working on non-assigned tasks.

If at any time you are working on a ticket and you realize it will take longer than originally estimated, bring it to the attention of the Traffic Organizer. He or She will then determine if tickets need to be shifted to maintain project timelines.

## Logging Hours

In addition to your normal Clock-In/Out, you will be recording the time spent on a given ticket.

While this will usually be development time, this includes (but is not limited to): research, computer setup, assisting a co-worker, debugging data issues, or general project management. This does not include lunch, breaks of any sort, or general chit-chatting with co-workers.

While your Clock-In/Out time is important, your logged hours will ultimately be used to evaluate your performance as a developer and your ability to keep up with the fast-paced environment we have. Failure to log your required hours each week may result in disciplinary action.

## Developer Environment

All development environments are set up the same. Trunks are setup to mimic production, with an individual database unique to each developer. Each developer will have the same appropriate tools installed as listed below for their specific operating system. This is done to allow any developer to work on another developers’ computer for any reason.

* Windows
  + PhpStorm - Editor/IDE
  + Notepad++ - Simple Text Editor
  + SQLYog – DB Connections
  + Docker – Testing
  + Slack - Chatting
* OS X
  + PhpStorm - Editor/IDE
  + TextWrangler – Simple Text Editor
  + Sequel Pro – DB Connections
  + Docker – Testing
  + Slack – Chatting
* Ubuntu
  + PhpStorm – Editor/IDE
  + GEdit – Simple Text Editor
  + MySQL Workbench – DB Connections
  + Docker – Testing
  + Slack – Chatting

## Master ID Export

We understand that some issues may not be able to be resolved without access to the actual data on the specific client to work with. We have developed an in-house tool at [www.deletedurlofthecompany.com](http://www.deletedurlofthecompany.com) to assist with pulling an up-to-date copy of any specific client that you may need for a ticket.

This tool works by automatically pulling any table that includes the column ‘masterid’. In addition, to these, several tables are included that all records in them. Any tables that aren’t included by these two parameters must be manually be added to the export script based on the proper query.

Exports are done one at a time in a queue-based method. Once the export is complete, it will be saved as an SQL file that you can download to re-upload at any time. Additionally, as part of the configuration, you may have it automatically deploy the changes to your specific trunk database.

## Developer Forks

When working on the code, developers will have their own ‘Forks’ of the repository. This fork is their own playground that can be used to house any changes and/or projects the developer works on. While the version control software will typically keep this clean for the remote repository, it is the responsibility of the developer to clean up their local branches on a regular basis. This includes removing merged, dead, or discontinued branches.

Please note that when developers are collaborating on projects, they will be able to see all the branches of the co-worker they are working with. As a result, the developer should always follow the Branching schema (See Branching) to ensure a clean, easy to manage, environment.

## Developer Collaboration

You may find that you will occasionally be required to collaborate on projects with another developer. In such a case, one of the developers will be the project lead. All code changes will go to their repository. In this case, be sure to not utilize rebasing and instead use merging.

## Branching

When working on tickets, each ticket must be done as its own branch. If a branch contains multiple tickets (and/or changes), it will delay the time it takes to merge the change in. See the ‘Additional Notes’ document for more details on why this is done.

Branch names should be kept uniform, include the parent branch, the ticket number, and a brief description of the changes to be made:

* Version-7.4-123456-new-awesome-feature
* v7.4-hf-broken-link
* broken-query-654321-v74

There is not specific format so long as it meets the above requirements and can be easily read by other developers.

## Commits

You should be committing frequently. It is suggested that you commit your code at any point in which the code changes you are working on can be classified as ‘working as intended’. This makes tracking changes and merge requests easier, as well as gives you a point in which you can backtrack your changes if something stops working.

You may put whatever information in your commits you deem appropriate, so long as it includes the ticket number and it is descriptive enough to describe the intent of the change. If at any time you do not know what to put as the commit message, it is suggested to use the phrase ‘Posterity Commit’.

It must be noted that when your commits are merged into the master repository, all your commits will be ‘squashed’ and turned into a single commit. This holds the purpose of keeping the code clean and easier to manage.

## Commit Signing

All commits should be signed using a registered GPG key. This is done as a security measure to ensure that all changes are made by the person whom authored the commit. The code repository will reject any commits that aren’t properly signed.

Please note that when doing a rebase, you will need to resign any commit you have made. Additionally, be aware that signed commits must be done through the command line unless your computer is properly configured to allow you to do this through your IDE.

## Automated Testing (Codeception)

We utilize Codeception for our automated testing. In addition to the ‘Unit Testing’ suite, Codeception also runs the ‘Functional Testing’ suite (Simulates a browser request), and ‘Acceptance

Testing’ suite (Runs tests against an actual browser). We also have a few in-house suites that are ran as well. These are all used to ensure that the code is always functioning as expected. It is your responsibility to add all appropriate testing for any work you do (See the attached ‘Codeception’ document for additional usage details).

The automated testing has only recently been adopted. Many parts of the system are legacy in nature and do not align well with Unit or Functional Testing and can only have acceptance testing. Any new classes must have unit testing and MVC modules must have functional testing. All new systems must have acceptance testing.

At the time of writing, our testing only covers 0.15% of the codebase. In an effort start increasing this number, you should always add acceptance testing to any legacy work you do. This testing need only test the one specific change you made and does not need to cover any more than the minimum required to get the testing running. For example, if you add a new field to the customer edit screen, and no testing yet exists for that screen, you will add the testing for just your new field. When running the testing, you may get additional database errors that have to be resolve by filling out additional fields on the form.

All testing will be automatically run through the Pipelines (see Pipelines). Please ensure that all testing is passing locally before pushing code your fork. Otherwise you may overburden the pipelines and cause them to take longer than normal.

## Quality Assurance Tools

In addition to the automated testing, we also utilize several tools to help ensure the quality of the code base. These tools include (but are not limited to) PHPStan, PHP-CS-Fixer, Infection, and Editor Config. These tools may run through the pipeline but can (and should) be ran on the developers’ local computer.

PHPStan is a static analysis tool. It will search the code for potential bugs/errors without the need to run the actual code. It will search for errors like Undefined Variable, Unknown Class, or other more detailed potential errors. We can extend this tool to search for specific errors and or/business rules we wish to enforce in the code base.

PHP-CS-Fixer is a tool to ensure that a file conforms to our established code format. It fixes/identifies items like proper spacing/tab characters, syntax formats like if statements with brackets, and many other code formatting items. This tool helps ensure that everyone’s base code is the same.

Infection is a mutation testing framework. It works by running the entire testing suite to determine code coverage. After code coverage is determined, Infection will modify any code that was covered and run the testing again. If your tests are accurate enough, they will fail and the mutant is considered squashed. If the testing passes, it is considered an escaped mutant and your testing may not be accurate enough. Infection testing will make changes to the code such as turning `++` to `--` or

`return $this` to `return null`. Note that Infection will not make changes that go against PHP’s return

type declarations.

Editor Config is more of a standard than a tool. It is used to ensure that every developers’ editor

is configured the same with details like using 4 spaces in place of tabs, aligning array elements, etc.

## Merge Requests (MR)

Once you have finished the changes to the code, completed all testing, and ran any QA tools, you should immediately fetch any recent changes from upstream and rebase your branch on upstream parent branch (See attached document on Git). Resolve any conflicts as required and ensure all testing/tools are still passing. You may then push the branch to your fork (A forced push may be required, see attached document on Git).

After you push the branch to your fork, you will create a merge request. You may also hear the

term ‘Pull Request’ or ‘PR’ (The equivalent names in GitHub). The MR should be named in the format ‘WIP:[XA][XB] XC XD’ where:

* WIP: Only put this if the merge request is still being worked on and you don’t want it merged in yet. This is useful if you want the opinion of another developer.
* XA: Numeric Version Number (IE: 7.1, 7.2)
* XB: Ticket Number (IE: 123456)
* XC: Describe the Intent of the Ticket (IE: Adding the Ability to Sort Locations by Name)
* XD: Any Additional ‘Helper’ Tags. IE:
  + Resolve rb#1234 – Resolves Rollbar Issue #1234 Automatically
  + Fixed – Notifies YouTrack to change the ticket to ‘Fixed’ Status

Using the above examples, a valid title would be `[7.1][123456] Adding the Ability to Sort Locations By Name Resolve rb#1234`

As your description, you must put the details of all areas of the code base that is touched as well as any alterations to the database. Additionally, add any relevant information the testing team can use to help speed up the QA process (See Testing/QA).

Please note that it is your responsibility to ensure that the Pipeline is passing (See Pipeline). If the Pipeline is not passing, your code will not get reviewed, tested, or merged. After the pipeline is passing, update the ticket to a status of ‘Awaiting Review’.

## Labeling Merge Requests

We utilize certain labels (AKA Tags) in the merge request process to help identify what step the MR is at or to identify any special cases. Common labels include:

* Alters Tables – Changes the Database
* Awaiting Merge – Changes approved by the QA team and the ticket is ready to merge.
* ?? Review After QA/External Review – Changes have to be reviewed by a third-party outside the QA team.
* Composer – Changes have been made to Composer Dependencies
* Duplicate – Duplicate MR/Ticket
* Enhancement – Flags as a new feature/enhancement
* ENV – Changes to the environment needed (IE: Nginx Settings)
* Rollbar – Fix for a rollbar issue.

When you create the merge request, be sure to apply any appropriate tags to the MR. Failure to do so could result in code getting merge in prematurely.

## Pipeline (Continuous Integration - CI)

We utilize a continuous integration platform often referred to as the ‘Pipeline’. The Pipeline takes care of automating several aspects of the codebase, including running the automated testing, building the code for production, and generating various statistics for quality assurance purposes.

Whenever your push to your fork, the Pipeline will automatically start running for the most recent commit. If a System Job for the Pipeline fails, the Pipeline will run the Job again up to a total of three runs. If the normal retry does not work, you are able to manually retry the Job. System Jobs are ones that the Pipeline uses to prepare for processing (IE: The `composer install` Job). Non-System Jobs are only run once. Each of the Automated Testing jobs are considered Non-System.

While not often, the Pipeline will occasionally fail with the message ‘Is Docker Running?’ (or similar). These errors are not associated with the job specifically and will usually clear themselves on the retry. If such errors do occur, and a retry does not resolve the issue, report it to the VP of Application Development.

## Code Review

After the Pipeline for your MR is passing, your changes will get reviewed by another developer or the VP of Application Development. If they catch anything that goes against our established procedures or the believe it can be done more efficiently, they will open a discussion on the item in question. It is your responsibility to work with the reviewer and resolve the discussions in a reasonable manner. If you are unable to resolve the discussion, report to the VP of Application Development. He or She will make the appropriate decision to resolve the discussion.

Once all discussions have been resolved on the Merge Request, update the ticket to have a

status of ‘Awaiting QA Review’.

## Testing/QA

After all discussions have been handled in the MR, the ticket will go to the QA department. They will pull the new changes down to their local computers and test the changes to ensure that the changes

accurately represent the intent of the ticket. Please note that the QA department is trained specifically to not touch any ticket where the pipeline is failing.

If they feel that the ticket does not, or the come across any bugs and/or issues with the changes, they will assign the ticket back to you detailing what is still to be done or what went wrong. They will also be keeping an eye out for changes in the quality or anything that goes against our established design standards. This may include (but is not limited to): consistency of field length, proper colors, validation checks, etc.

Once the QA team has signed off on the changes, the ticket is ready to be merged. What happens with a specific ticket will depend on which version of the software the ticket is for.

## Merging Code

Merging code depends on the upstream version of the code that the changes are for: Stable or

Beta/Alpha. The QA team’s permissions in the repository will enforce these procedures.

If the changes are applied to the beta and/or alpha version, the QA team has the authority to go ahead and merge the changes, provided there are not table alterations to be made.

If the changes are for the stable version of the code, or if the MR is flagged as ‘Altering Tables’, the QA is not authorized to do the merge. In such case, the VP of Application Development will merge the code changes in. This is typically done twice a day, once in the early morning, and once shortly after noon.

## Deploying Code

Deployment of code is also based off the version of the code the change applies to. If the change applies to the Stable version of the code, it is done manually through the pipeline process by the VP of Application Development. If the change applies to the Beta or Alpha branches, code will get automatically deployed through the pipeline once the pipeline passes.

## Refactoring Code

All code refactors must be done by a ticket labeled specifically ‘Code Refactor’. At no point should code be refactored as part of a bug fix, feature addition, or tweaks to the code. If at any point you feel that the code should be refactored, create a new ticket separate from the one you are working

on. The ticket will be assigned to the ‘Code Refactor’ project and severity. Refactoring code outside of a

code refactor ticket may result in disciplinary action.

If you are working on a ticket and you feel that ticket requires a refactoring, but the ticket is not a code refactor, bring it immediately to the attention of the VP of Application Development. The decision will then be made to either allow the refactor, do the refactor as a separate ticket later, do the refactor as a precursor to the ticket, or deny the code refactor in full.

## Slack

We utilize the ‘Slack’ software for communication purposes. This software allows both user-to- user communication as well as group communication. Every developer is required to create an account using the company email address. All developers must have Slack open. In the event another developer or management needs to ask you a quick question that does not go through emails, they will ‘Slack’ it to you instead. This is done to ensure that questions can be asked without completely breaking your attention as a developer while in the middle of a project.

## Rollbar

In addition to our ticket system, we utilize the Rollbar software to track errors in the codebase. Whenever an error occurs, the system will automatically record the error to Rollbar. Rollbar tracks stats like when errors occur, how frequently they occur, who they occurred for.

Rollbar will automatically create a ticket in Mantis. If an error occurs repeatedly, the severity of the ticket will automatically be elevated. Most tickets from Rollbar will be handled by the Break Fix team.

## Make (Makefile)

We utilize ‘make’ to assist with common repetitive tasks. It is also used for tasks like preparing your environment for testing and/or trunk development, running docker images, running the QA tools or other such tasks. All tasks may be run by using the terminal command ‘make [job]’ where [job] is (but not limited to):

* env-trunk – Initialize your trunk environment file
* env-testing – initialize your testing environment file
* test – Run all test suites
* test-unit – Run just the unit test suite
* test-acceptance – Run docker and the acceptance suite.

You may look at the file named ‘Makefile’ in the root of the site to see all available jobs.

Information on properly reading the Makefile is readily available online.

## Naming Conventions - General

We always use Lower Camel Case as our primary naming conventions. This includes for method/function names, variables, and database column names. All variables should be self- documenting, but not overly so:

* Good: employeeModel
* Bad: getPolicyEmployeeModel

## Naming Conventions - Classes

Classes should all be named in PascalCase. This includes their namespace as well. We utilize the PSR-4 standard. All Classes should be in a namespace that matches the directory structure. The

exception to this is the first level, which should be ‘Matrix’. All new classes should be under the amapi/app/modules/ Folder. The file name should match the name of the class and end with ‘.php’.

## Coding Conventions - S.O.L.I.D., D.R.Y., K.I.S.S.

All classes should follow the industry standards of OOP development. Here we’ve collected several examples for you to follow: https: www.deletedcompanyurlonpurpose.com

## Coding Conventions - Single Responsibility Principle (SRP)

SRP is covered by the SOLID coding principles but warrants extra mention. Any class and/or

method should do one thing only. A good example of a bad example of this is the ‘GenericDropdown’ method of the ‘DropDownClass’ class. While an exteremely powerful method, this method does several different things, including (but not limited to): build query, query results, cache results, build select html, iterate over results. If we were to make a ‘MobileDropDown’ class and extend DropDownClass, we

would have to override the full method to be able to properly extend it. This creates duplicate code that just makes the code harder to maintain.

SRP can also apply to traditional functional code. You will often see instances of code like ‘if (A && B) {} else if (A && C) {} elseif (A && D)’. Not the repeated first variable.

## Coding Conventions - Breaking Up Functionality

The larger your class is, the harder it is to maintain and unit-test. We have some classes that have over 6000 lines of code. Break functionality up into additional helper classes before allowing your class to get this large.

### Ticket Work Flow

PM

YES

Start

Open

Unassigned Ticket

Status = New

Reporter

Needs

Design

Review ticket

BA

Create Wireframe &

or Written Specs

Assign to BA

Status = Confirmed

Assign to PM

Status = Confirmed

Developer

NO

YES

Immediate

Assign Ticket to

Developer Status = Confirmed

PM

Pause Current Ticket Status = Paused

[To](#_bookmark0)

[Main 2](#_bookmark0)

NO

### Ticket Work Flow

QA

QA

* Developer should follow curr ent assignment ru les

Review

Ticket

Begin Working on

Ticket

Status = In Progress

Complete Ticket & Assign to Sr.

Developer

Status = Awaiting

Review

* Developer will review specs and/ or wireframe if applicable and work with BA and/ or User to review design questions

[From](#_bookmark0)

[Main 1](#_bookmark0)

Developer

Review Code

|  |  |
| --- | --- |
| BA | |
| Test the Requirements | |
|  |  |

Feedback

End

Update & Close Ticket

Status = Closed

Reporter

Sr. Developer

YES

Feedback

NO

* Code Merged an d Deployed to the versions in dicate d on ticket

YES

Feeback

YES

NO

NO

* Table change s Deployed to the versions in dicate d on ticket

NO

Successful

YES

Table

Changes

YES

NO

NO

YES

Feedback

NO

BA Review

Assign Ticket to QA

Status = Awaiting

QA Review

Test Code

Infrastructure

Alter Table

QA

Confirm Table Changes

Assign to Infrastucture Status = Awaiting Table Changes

Assign to BA

Status = Awaiting BA Review

Assign Ticket to QA Status = Table

Changes Scheduled

Test

Ticket

Status = Awaiting Merge

Merge & Deploy

Ticket

Assign Ticket to Reporter Status = Resolved

YES

Re-Assign Ticket Back to Developer Status = Feedback