FRM
Master Test Plan

Version <1.0>

Revision History

Date	Version	Description	Author
25/04/2017	1.0	Creation and filling with basic information	Karl Spickermann

Table of Contents

<Iteration/ Master> Test Plan

1. Introduction

1.1 Purpose

The purpose of the Iteration Test Plan is to gather all of the information necessary to plan and control the test effort for a given iteration. It describes the approach to testing the software, and is the top-level plan generated and used by managers to direct the test effort.

This *Test Plan* for the FRM supports the following objectives:

- Controller
- Model
- View

1.2 Scope

Integration Testing with Travis CI and PHPUnit

- Travis CI for managing the testing by triggering builds and tests
- PHPUnit for coding the actual tests

Unit tests

- Will test the internal application logic.

Testing with end user

- Will test the user interface if it is easy to learn.

1.3 Intended Audience

- Students
- Professors
- Programmer

1.4 Document Terminology and Acronyms

n/a

1.5 References

[This subsection provides a list of the documents referenced elsewhere within the **Test Plan**. Identify each document by title, version (or report number if applicable), date, and publishing organization or original author. Avoid listing documents that are influential but not directly referenced. Specify the sources from which the “official versions” of the references can be obtained, such as intranet UNC names or document reference codes. This information may be provided by reference to an appendix or to another document.]

2. Evaluation Mission and Test Motivation

Testing is done to guarantee that the software is stable and furthermore stays stable over the development of new features and bug fixes.
[Provide an overview of the mission and motivation for the testing that will be conducted in this iteration.]

2.1 Background

By testing our project, we can monitor the effects that changes to the source code and user interactions cause to the functionality and performance of the software.

As a result we can:

1. Ensure that what we create does what it’s supposed to do.

   Testing guarantees that new functionalities work as intended and detects possible conflicts between the new and old functionalities. As an example a new feature could break an old legacy feature by testing we can prevent this from happening and safe our users from the trouble of dysfunctioning core services.

2. Catch all possible edge cases.

   “No user would ever do that.” This sentence creates edge cases. No developer can ever think of all possible combination of user interactions possible in his system to still catch all possible bugs ,hidden in bizarre action combination, excessive testing is needed.

2.2 Evaluation Mission

Testing is done to provide a stable software. And we will fulfill the goal by the following points.

- find as many bugs as possible
- find important problems
- certify to a standard
- verify a specification (requirements, design or claims)

2.3 Test Motivators

- Reduce quality and technical risks.
- Functional and no-functional requirements
- Design elements
- realize use cases faster by providing stability

3. Target Test Items

The listing below identifies those test items—software, hardware, and supporting product elements —that have been identified as targets for testing. This list represents what items will be tested.

- Controller (Logic)
- View (Design)
- Model (Database)
- Routing (Interaction of parts above)

4. Outline of Planned Tests

4.1 Outline of Test Inclusions

- Integration Testing with Travis CI
- Unit Testing with PHPUnit
- Testing with the end user

4.2 Outline of Other Candidates for Potential Inclusion

Stress testing the application

4.3 Outline of Test Exclusions

n/a

5. Test Approach

- Testing with end user
- Integration Test
- Unit Test

5.1 Initial Test-Idea Catalogs and Other Reference Sources

n/a

5.2 Testing Techniques and Types

Testing with end user

Technique Objective:	Testing the simplicity of the app
Technique:	● Testing the menu for simplicity ● Testing the app for easy understanding ● Users fill out a survey
Oracles:	The test users are happy with the app. The whole app is easy to understand, it is self-explaining. The menu navigation is simple.
Required Tools:	A Device capable of navigating and interacting with the website. (Preferable a Laptop or Desktop Computer)
Success Criteria:	The user is happy.
Special Considerations:	-

Integration Test

Technique Objective:	Testing if the combination of units work well together.
Technique:	• Whenever a feature branch merges with the master branch Travis CI automatically triggers a build and runs Unit Tests •
Oracles:	We assume, that all tests pass.
Required Tools:	CI tool – Travis CI
Success Criteria:	• 40% test coverage • all tests pass in deployment process
Special Considerations:	-

Unit Test

Technique Objective:	Testing the functionality of the code
Technique:	Testing the code of the testable classes.
Oracles:	We assume, that all tests pass.
Required Tools:	PHPUnit
Success Criteria:	All test pass
Special Considerations:	-

6. Entry and Exit Criteria

6.1 Test Plan

6.1.1 Test Plan Entry Criteria

This Test Plan can begin as soon as the development and build environment is set, all Use Cases are defined and the development has begun.

6.1.2 Test Plan Exit Criteria

With the successful deployment of a FRM-System on another server including a test of all its features the product can be labelled as fully functional and testing can end.

6.1.3 Suspension and Resumption Criteria

If the project gets cancelled testing would stop prematurely.

6.2 Test Cycles

6.2.1 Test Cycle Entry Criteria

n/a

6.2.2 Test Cycle Exit Criteria

n/a

6.2.3 Test Cycle Abnormal Termination

n/a

7. Deliverables

7.1 Test Evaluation Summaries

Travis runs the functional and unit tests on each push. It can either fail, pass or error.

7.2 Reporting on Test Coverage

We use Coverall.io to publish our code coverage information: https://coveralls.io/github/d-wagner/frmsystem

7.3 Perceived Quality Reports

With the help of phpmetrics we are able to present an overview of the quality of our codebase.

7.4 Incident Logs and Change Requests

N/A

7.5 Smoke Test Suite and Supporting Test Scripts

n/a

7.6 Additional Work Products

n/a

7.6.1 Detailed Test Results

n/a

7.6.2 Additional Automated Functional Test Scripts

n/a

7.6.3 Test Guidelines

n/a

7.6.4 Traceability Matrices

n/a

8. Testing Workflow

We use PHPUnit for testing as our build process supports testing, every push to the master branch causes the functional and unit tests to be run by Travis CI. All the test results are reported to Coveralls.io.
If the build and all quality reports were successful the project is deployed to our server. If a build failed the creator of the build is notified to ensure that the issues are addressed immediately.
We r perform end user tests when we think it is necessary.

9. Environmental Needs

[This section presents the non-human resources required for the **Test Plan**.]

9.1 Base System Hardware

The following table sets forth the system resources for the test effort presented in this *Test Plan*.

System Resources		
Resource	**Quantity**	**Name and Type**
Database Server	1	SERVER4YOU, vServer SSD S8
Test Environment	1	localhost, xampp

9.2 Base Software Elements in the Test Environment

The following base software elements are required in the test environment for this *Test Plan*.

Software Element Name	Version	Type and Other Notes
Ubuntu	16.04	Operating System
Apache	newest	Web Server
Chrome	newest	Internet Browser
PHP Storm	7.0.6	IDE
Node.js	6.2.0	Serversided Script

Software Element Name	Version	Type and Other Notes
Travis CI		CI Enviroment

9.3 Productivity and Support Tools

The following tools will be employed to support the test process for this *Test Plan*.

Tool Category or Type	Tool Brand Name	Vendor or In-house	Version
Metrics	phpmetrics	Open Source	newest
Testing	PHPUnit	Open Source	5.3
Test Coverage Monitor or Profiler	Coveralls.io	Coveralls	newest
Project Management	Youtrack	Jetbrains	newest
DBMS tools	MySQL	Open Source	newest

9.4 Test Environment Configurations

The following Test Environment Configurations needs to be provided and supported for this project.

Configuration Name	Description	Implemented in Physical Configuration
Average user configuration	Number of users accessing the application at the same time	1
Minimal configuration supported	Speed and power of the internet connection provided by the server host.	100Mbit/s
Network installation (not client)	Performance of the application server and database server.	WebServer with 2GB Ram and Ubuntu 16.04

10. Responsibilities, Staffing, and Training Needs

10.1 People and Roles

This table shows the staffing assumptions for the test effort.

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
Test Manager	1	Provides management oversight. Responsibilities include: • planning and logistics • agree mission • identify motivators • acquire appropriate resources • present management reporting • advocate the interests of test • evaluate effectiveness of test effort
Test Analyst	2	Identifies and defines the specific tests to be conducted. Responsibilities include: • identify test ideas • define test details • determine test results • document change requests • evaluate product quality
Test Designer	2	Defines the technical approach to the implementation of the test effort. Responsibilities include: • define test approach • define test automation architecture • verify test techniques • define testability elements • structure test implementation
Tester	3	Implements and executes the tests. Responsibilities include: • implement tests and test suites • execute test suites • log results • analyze and recover from test failures • document incidents

Human Resources		
Role	**Minimum Resources Recommended** **(number of full-time roles allocated)**	**Specific Responsibilities or Comments**
Test System Administrator	1	Ensures test environment and assets are managed and maintained. Responsibilities include: • administer test management system • install and support access to, and recovery of, test environment configurations and test labs
Database Administrator, Database Manager	1	Ensures test data (database) environment and assets are managed and maintained. Responsibilities include: • support the administration of test data and test beds (database).
Designer	1	Identifies and defines the operations, attributes, and associations of the test classes. Responsibilities include: • defines the test classes required to support testability requirements as defined by the test team
Implementer	3	Implements and unit tests the test classes and test packages. Responsibilities include: • creates the test components required to support testability requirements as defined by the designer

10.2 Staffing and Training Needs

This section outlines how to approach staffing and training the test roles for the project.

n/a

11. Iteration Milestones

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
At least 20% Test Coverage	Project Start	Project Start	30.6.2017	
Release Build Passes	Project Start	Project Start	30.6.2017	

12. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Technical Problems	<Tester> needs to be informed and has to fix problems.	• Fix the problem
Test data proves to be inadequate.	<Customer> will ensure a full set of suitable and protected test data is available. <Tester> will indicate what is required and will verify the suitability of test data.	• Redefine test data • Review Test Plan and modify • components (that is, scripts) • Consider Load Test Failure
Database requires refresh.	<System Admin> will endeavor to ensure the Database is regularly refreshed as required by <Tester>.	• Restore data and restart • Clear Database

Dependency between	Potential Impact of Dependency	Owners

Assumption to be proven	Impact of Assumption being incorrect	Owners

Constraint on	Impact Constraint has on test effort	Owners

13. Management Process and Procedures

n/a

13.1 Measuring and Assessing the Extent of Testing

n/a

13.2 Assessing the Deliverables of this Test Plan

n/a

13.3 Problem Reporting, Escalation, and Issue Resolution

n/a

13.4 Managing Test Cycles

n/a

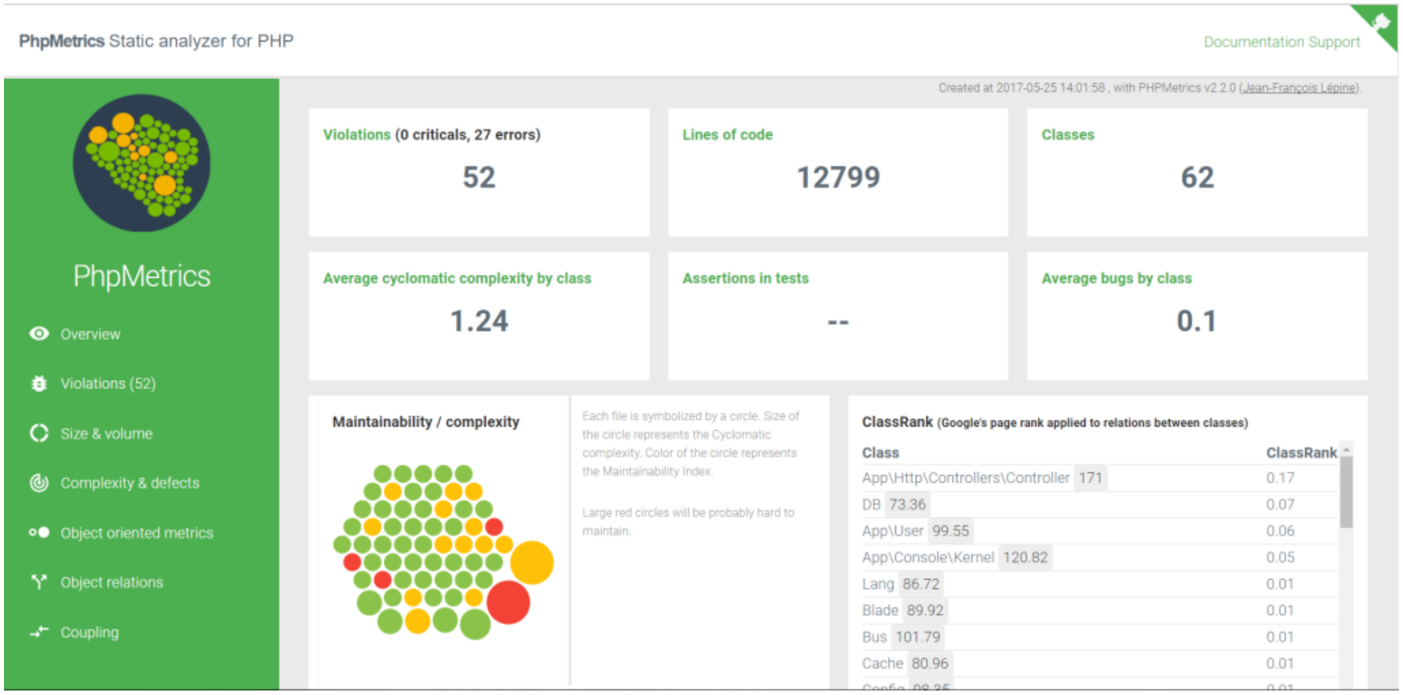
13.5 Traceability Strategies

n/a

13.6 Approval and Signoff

n/a

Appendix – Metrics Reports



PhpMetrics Static analyzer for PHP

Documentation Support

Created at 2017-05-25 14:01:58 , with PHPMetrics v2.2.0 (Jean-François Lépine).

PhpMetrics

- Overview
- Violations (52)
- Size & volume
- Complexity & defects
- Object oriented metrics
- Object relations
- Coupling

Violations (0 criticals, 27 errors): 52

Lines of code: 12799

Classes: 62

Average cyclomatic complexity by class: 1.24

Assertions in tests: --

Average bugs by class: 0.1

Maintainability / complexity

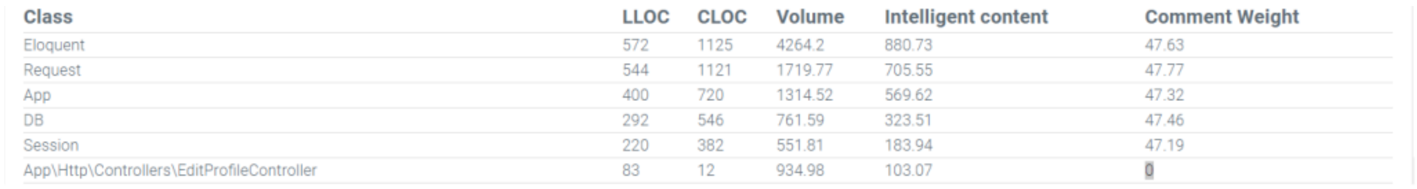
Each file is symbolized by a circle. Size of the circle represents the Cyclomatic complexity. Color of the circle represents the Maintainability Index.

Large red circles will be probably hard to maintain.

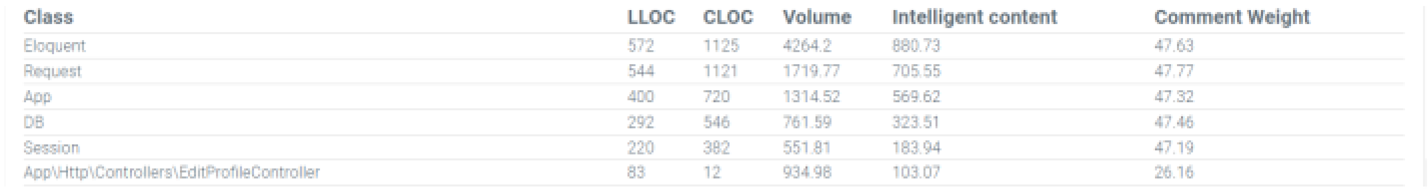
ClassRank (Google's page rank applied to relations between classes)

Class		ClassRank
App\Http\Controllers\Controller	171	0.17
DB	73.36	0.07
App\User	99.55	0.06
App\Console\Kernel	120.82	0.05
Lang	86.72	0.01
Blade	89.92	0.01
Bus	101.79	0.01
Cache	80.96	0.01

Appendix – Comment Weight

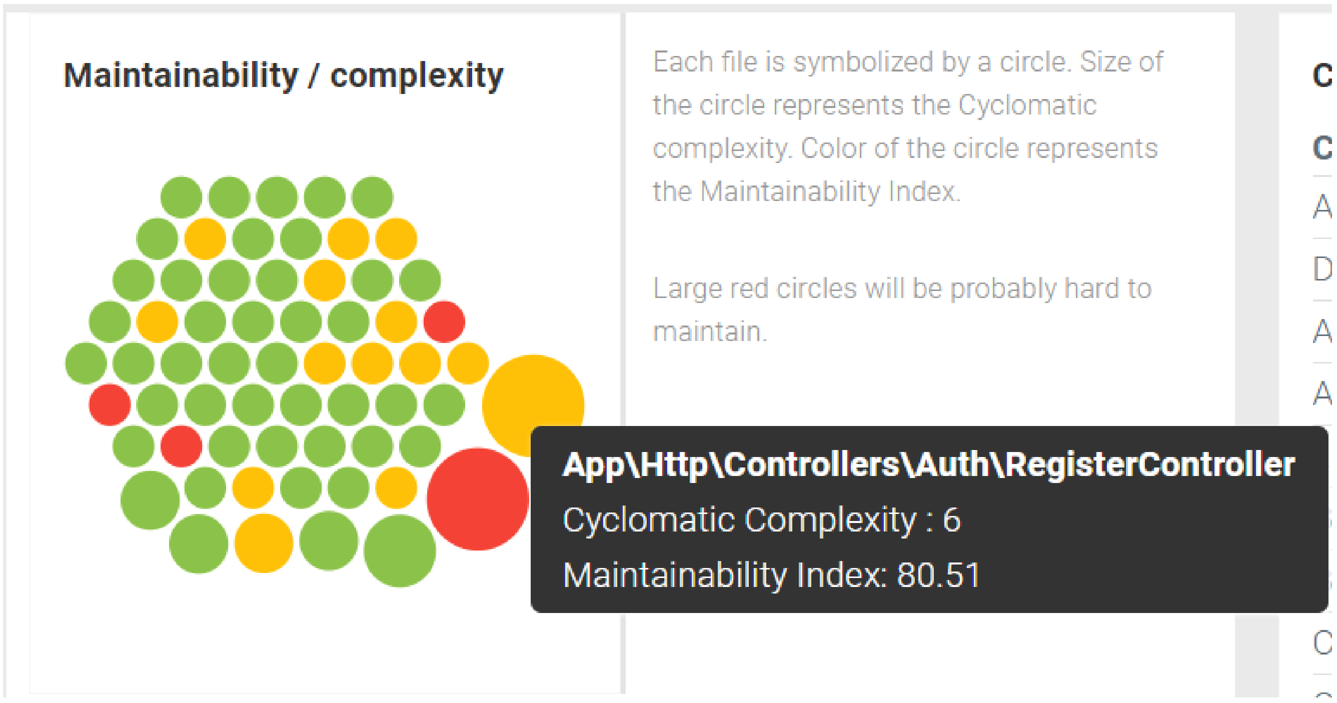


Class	LLOC	CLOC	Volume	Intelligent content	Comment Weight
Eloquent	572	1125	4264.2	880.73	47.63
Request	544	1121	1719.77	705.55	47.77
App	400	720	1314.52	569.62	47.32
DB	292	546	761.59	323.51	47.46
Session	220	382	551.81	183.94	47.19
App\Http\Controllers\EditProfileController	83	12	934.98	103.07	0



Class	LLOC	CLOC	Volume	Intelligent content	Comment Weight
Eloquent	572	1125	4264.2	880.73	47.63
Request	544	1121	1719.77	705.55	47.77
App	400	720	1314.52	569.62	47.32
DB	292	546	761.59	323.51	47.46
Session	220	382	551.81	183.94	47.19
App\Http\Controllers\EditProfileController	83	12	934.98	103.07	26.16

Appendix – Cyclomatic Complexity



Maintainability / complexity

Each file is symbolized by a circle. Size of the circle represents the Cyclomatic complexity. Color of the circle represents the Maintainability Index.

Large red circles will be probably hard to maintain.

App\Http\Controllers\Auth\RegisterController
Cyclomatic Complexity : 6
Maintainability Index: 80.51