FRM-System Blog

MENU

Final Blog Post

Please keep in mind: The blog of the first semester somehow got lost, Bplaced just deleted our account. Because of that only the blogposts of the second semester can be referenced here.

We realized the table as word to make it easier to navigate, here is the link: final blog post

As an archive, please use these 2 links: link, link

21. June 2017 / Leave a comment

Semester 2, Week 10: Continous Integration

Hi guys,

this will be a fairly short post.

Our CI-setup has not changed much since last time. Only codacy was added, which is triggered through every git commit and analyzes our code every time we change something. We also use phpmetrics for manual code analyzation from time to time but Codacy is the tool that we rely on most for the metrics.

For the installation part please have a look at the last post.

best regards,

Daniel W.

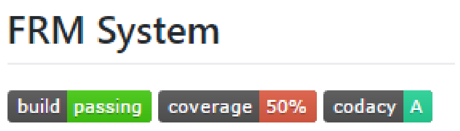
19. June 2017 / Leave a comment

Semester 2, Week 9: Testing 2

Hi guys,

This weeks post is about the testing procedure again. In an earlier post we already explained most of the stuff, so this is just a little update because the test document is now finished. For detailed information on how we test have a look at this first post regarding this topic.

Our badges (can be found here) are currently looking like this:


FRM System

build passing | coverage 50% | codacy A

The reason for the poor test coverage is, that we simply do not find the time to write tests for all the latest code additions that we made over the last 2 weeks. I think we demonstrated well enough how good of a test coverage we can achieve by having over 90% coverage before the coding of the last weeks.

Our finished test plan can be found here. We incorporated a load test of our server by now as well.

Also, we finished the installation tutorial that can be found in the readme of the Git repo. It is fully working and an installation test was carried out by David who will confirm this in a comment under this post.

best regards,

Daniel W.

19. June 2017 / Leave a comment

Semester 2, Week 8: Metrics

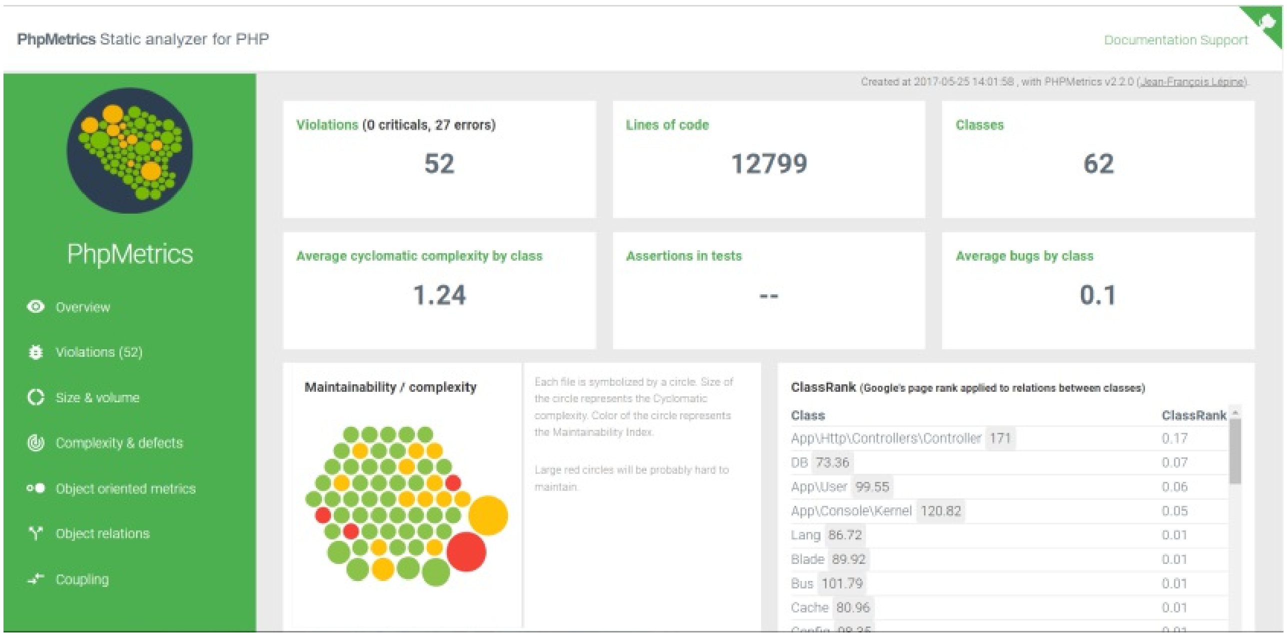
Hi guys,

this week is all about metrics.

Metrics are used to display specific aspects of a business, mostly in form of numbers. In Software Engineering this aspects are mostly connected to the code of programs.
The clear definition of metrics in software engineering is defined in the IEEE Standard 1061 from 1998: “A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.”

The outcome of a metric can be used to either further develop the project or correct bugs and possibly threats in the code base.

For this week we needed to collect our own measurements, for which we used the open source tool “phpmetrics”. This static analysis tool scans the php file of a project and creates an interactive HTML report of a group of metrics.



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

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.

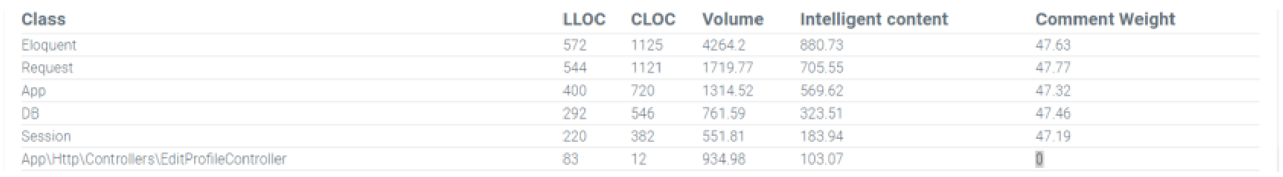
Large red circles will be probably hard to maintain.

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

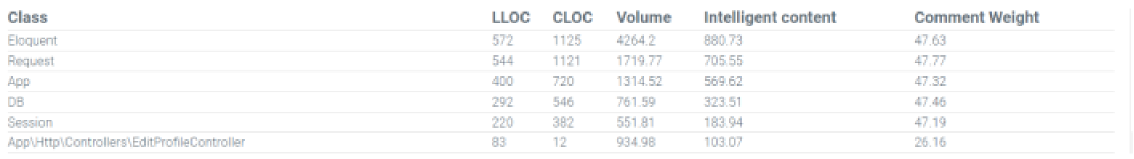
With the help of phpmetrics we could find a few useful measurements but for this blog entry we will concentrate on two of them Comment weight and Cyclomatic complexity.

Comment weight measure the ratio between logical code and comments. It is one of the few metrics where one can’t say that higher or lower is always better. Comment weight needs to be in the middle, depending on the complexity of the code comment weight should be between 10 and 30 percent.



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

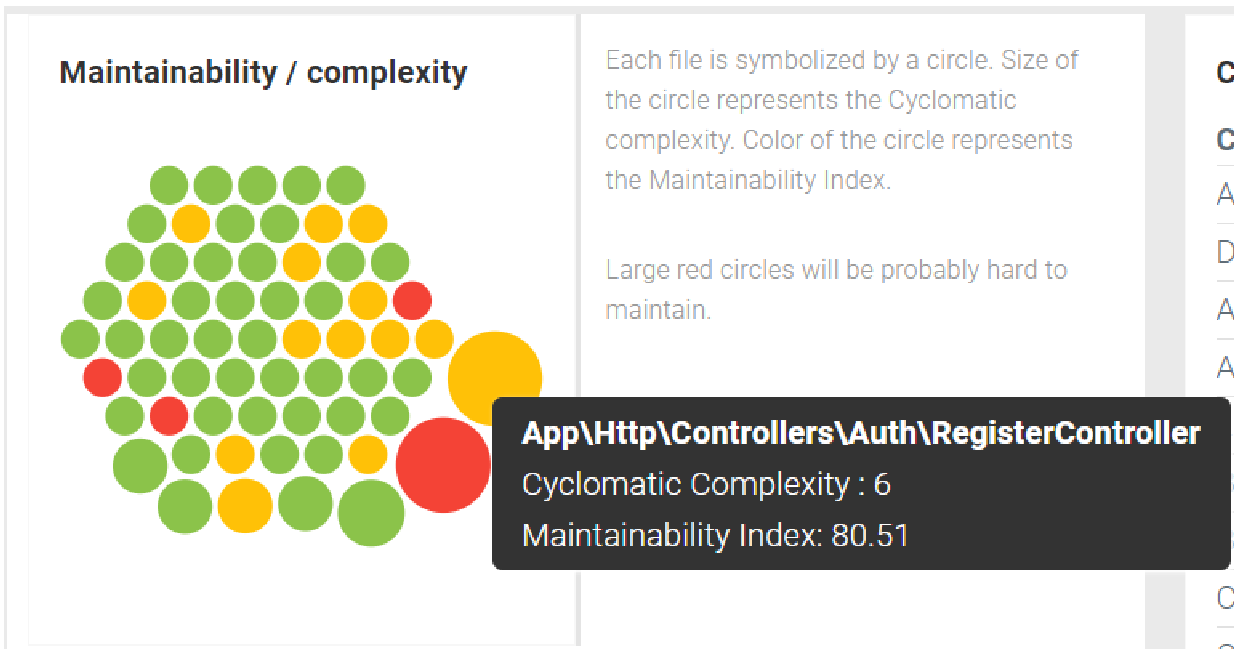
If you look at the Comment weight for our project you see that we have a class with a measurement of zero.



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

To fix our metrics we will simply add a few comments to the basic functionality of this class.

Cyclomatic complexity indicates the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program’s source code. For instance, if the source code contained no control flow statements (as an example if), the complexity would be 1, since there would be only a single path through the code. If the code had one single-condition IF statement, there would be two paths through the code: one where the IF statement evaluates to TRUE and another one where it evaluates to FALSE, so the complexity would be 2.



When we look at our project measurements again, we see that phpmetrics flag one of our classes for high effort maintainability. The maintainability is calculated from the Cyclomatic complexity, Lines of Code and a Halstead Metric. The easiest way to lower the maintainability score would be to lower the Cyclomatic complexity. But since the flagged class is a automatic generated part of the PHP Framework Laravel, we abstain from changes to the file.

That should be it for this week. Feel free to leave comments and suggestions.

Best regards,

Karl Spickermann

7. June 2017 / 2 Comments

Semester 2, Week 7: Patterns and Refactoring

Hi there,

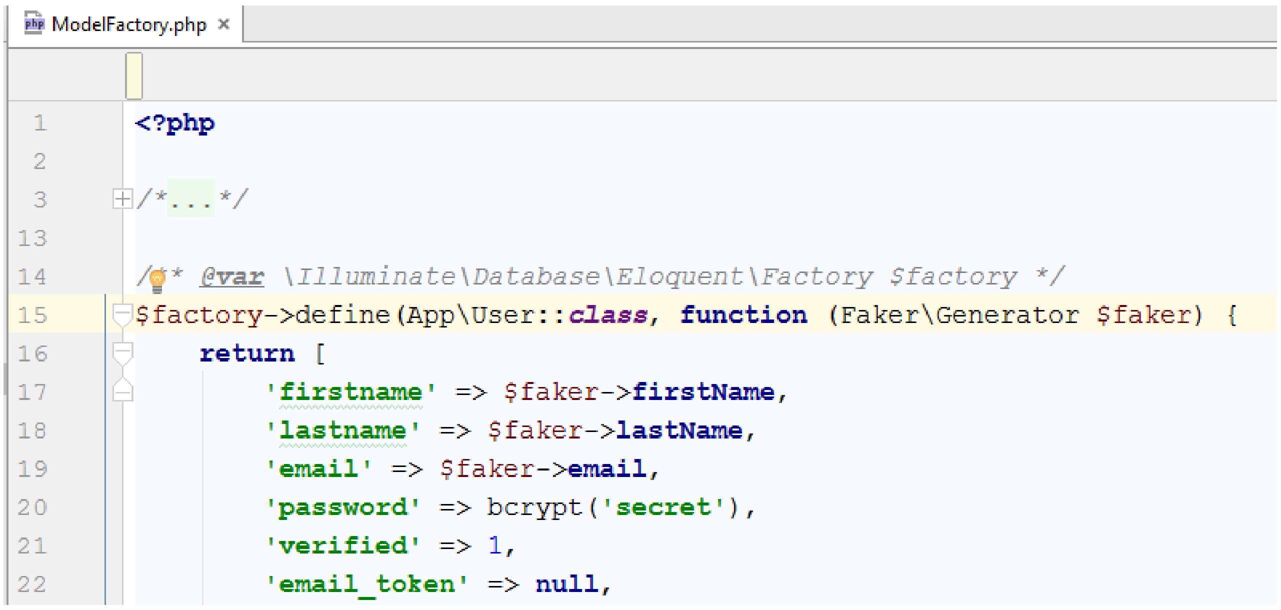
this weeks task produced a lot of headaches for us. We tried to find a way of including a pattern somewhere in the code, that does not make use of the patterns that the Laravel framework specifies. As far as we know, utilizing these does not really officially count towards the grading criteria.

If you had a look at our previous class-diagrams or some of our code, you might understand, that there is no obvious piece of code, that would benefit of other complex patterns than the ones provided by Laravel. If go by the Laravel documentation when writing your application, which we tried to do, everything is kind of clear how to do. Of course, you can do things in several other ways – even in Laravel. After all PHP is a very powerful language. But these ways are often even more complicated than using the desired way for that. The Laravel mechanics are sometimes hard to understand if you want to really know what you are doing, but they are very powerful and useful because everything is well thought through. If you use strange, unintended ways for realizing things, you might find yourself redoing everything at a later stage since other stuff that builds on top does not work with "hacky" solutions.

Because of that we decided to not implement a pattern that is plain stupid for what our application is doing. We are not firm enough to estimate if such an action might have extremely big negative sideffects on our other usecases that are yet to come. Instead, I focused on getting some more tests done, because that really highlights a pattern that laravel provides. Doing this, our testcoverage is now really satisfying, as can be seen by our current batch:

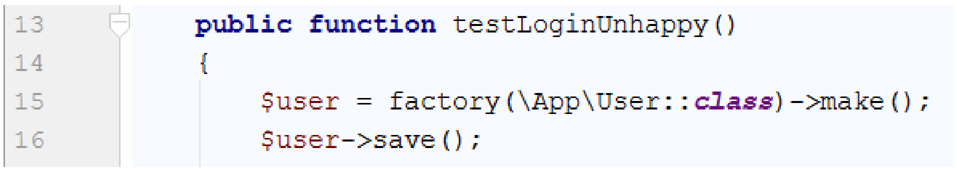
build passing coverage 95%

For testing you need lots of testdata. This is generated by faker, a php package, that generates data following usual data-patterns e.g. emails, names, dates etc. But it would be very tedious to generate a user or a relationship by hand everywhere you need it in your tests. This is avoided by so called factories, the pattern type is also named "factory method". Here you can see a snippet of our model factory declaration, we did this for all our models:



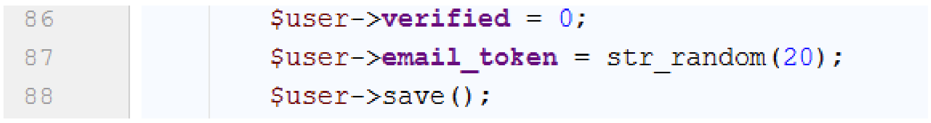
```php
<?php

/*...*/

/** @var \Illuminate\Database\Eloquent\Factory $factory */
$factory->define(App\User::class, function (Faker\Generator $faker) {
    return [
        'firstname' => $faker->firstName,
        'lastname' => $faker->lastName,
        'email' => $faker->email,
        'password' => bcrypt('secret'),
        'verified' => 1,
        'email_token' => null,
```

To utilize the factory, it is now just one line in your test, to get a user instance:



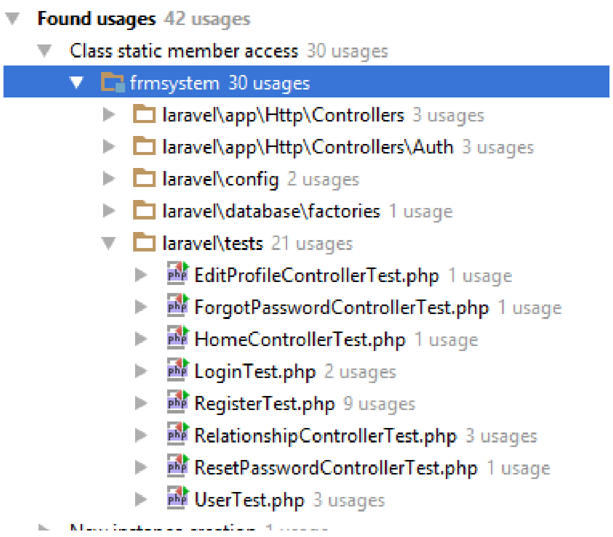
```
    public function testLoginUnhappy()
    {
        $user = factory(\App\User::class)->make();
        $user->save();
```

The great thing is, you can modify it to your tests needs before saving the model to the database. For a register test we needed an unverified user, this can be easily done as well:



```
        $user->verified = 0;
        $user->email_token = str_random(20);
        $user->save();
```

In the next screenshot you can see how much usage we get out of **the user-model-factory alone.** As described, the laravel mechanics are very useful, but getting everything working is sometimes not that easy…



```
▼ Found usages 42 usages
  ▼ Class static member access 30 usages
    ▼ frmsystem 30 usages
      ▶ laravel\app\Http\Controllers 3 usages
      ▶ laravel\app\Http\Controllers\Auth 3 usages
      ▶ laravel\config 2 usages
      ▶ laravel\database\factories 1 usage
      ▼ laravel\tests 21 usages
        ▶ EditProfileControllerTest.php 1 usage
        ▶ ForgotPasswordControllerTest.php 1 usage
        ▶ HomeControllerTest.php 1 usage
        ▶ LoginTest.php 2 usages
        ▶ RegisterTest.php 9 usages
        ▶ RelationshipControllerTest.php 3 usages
        ▶ ResetPasswordControllerTest.php 1 usage
        ▶ UserTest.php 3 usages
  ▶ New instance creation 1 usage
```

That should be it for this week. If we find a real usage for patterns in the other use cases we will of course change this post, but as far as we can tell, this is unlikely to happen.

Best regards,

Daniel W.

17. May 2017 / 3 Comments

Semester 2, Week 6: Refactoring exercise

Hi guys,

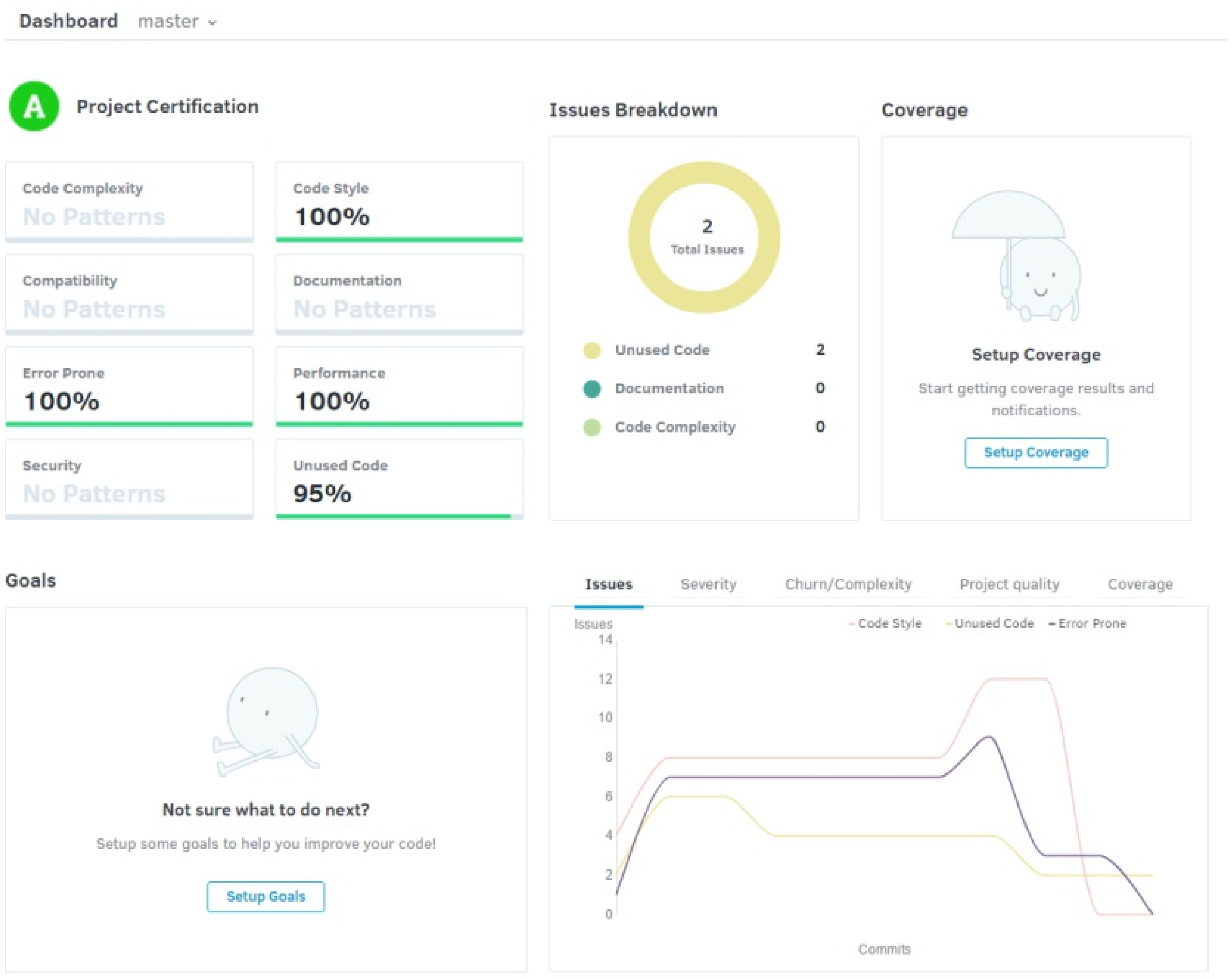
please view our repositories for the refactoring exercise here:

Daniel W.

Karl

Daniel R.

I also ran the result through Codacy, all the standard parameters are green. I also wrote tests for everything myself, they are running thorugh fine. (I did not configure Coverage on Codacy, feel free to clone the repo yourself and run all the tests in the test-package)


Dashboard master

Project Certification A

Code Complexity No Patterns	Code Style 100%
Compatibility No Patterns	Documentation No Patterns
Error Prone 100%	Performance 100%
Security No Patterns	Unused Code 95%

Issues Breakdown

2 Total Issues

●	Unused Code	2
●	Documentation	0
●	Code Complexity	0

Coverage

Setup Coverage

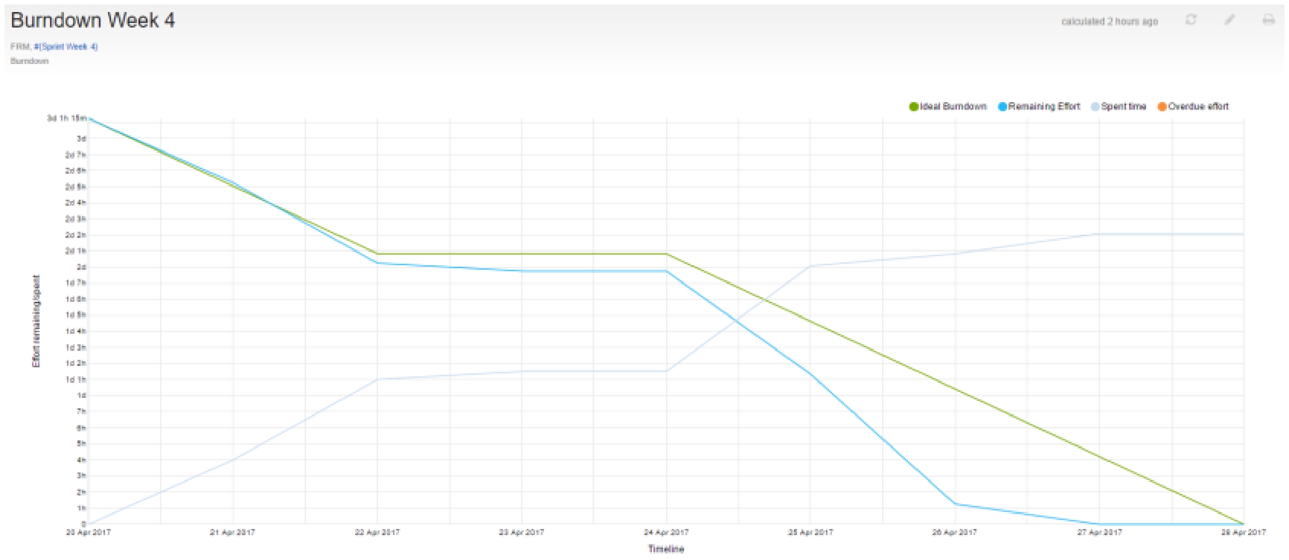
Start getting coverage results and notifications.

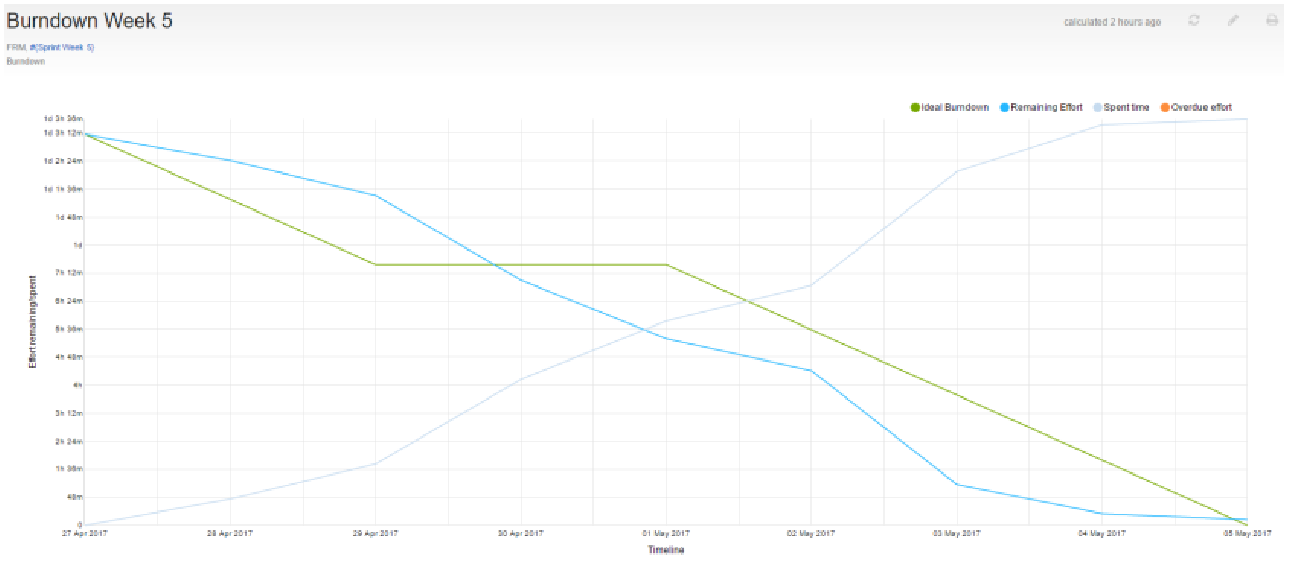
Goals

Not sure what to do next?

Setup some goals to help you improve your code!

It is also worth mentioning, that we now have achieved 2 kind of healthy sprints. Tracking with YouTrack is a lot easier and more convenient than with Jira, so our sprints are finally working out a little better.





Best regards,

Daniel W.

13. May 2017 / 2 Comments

Semester 2, Week 4: Continuous Integration and Testing

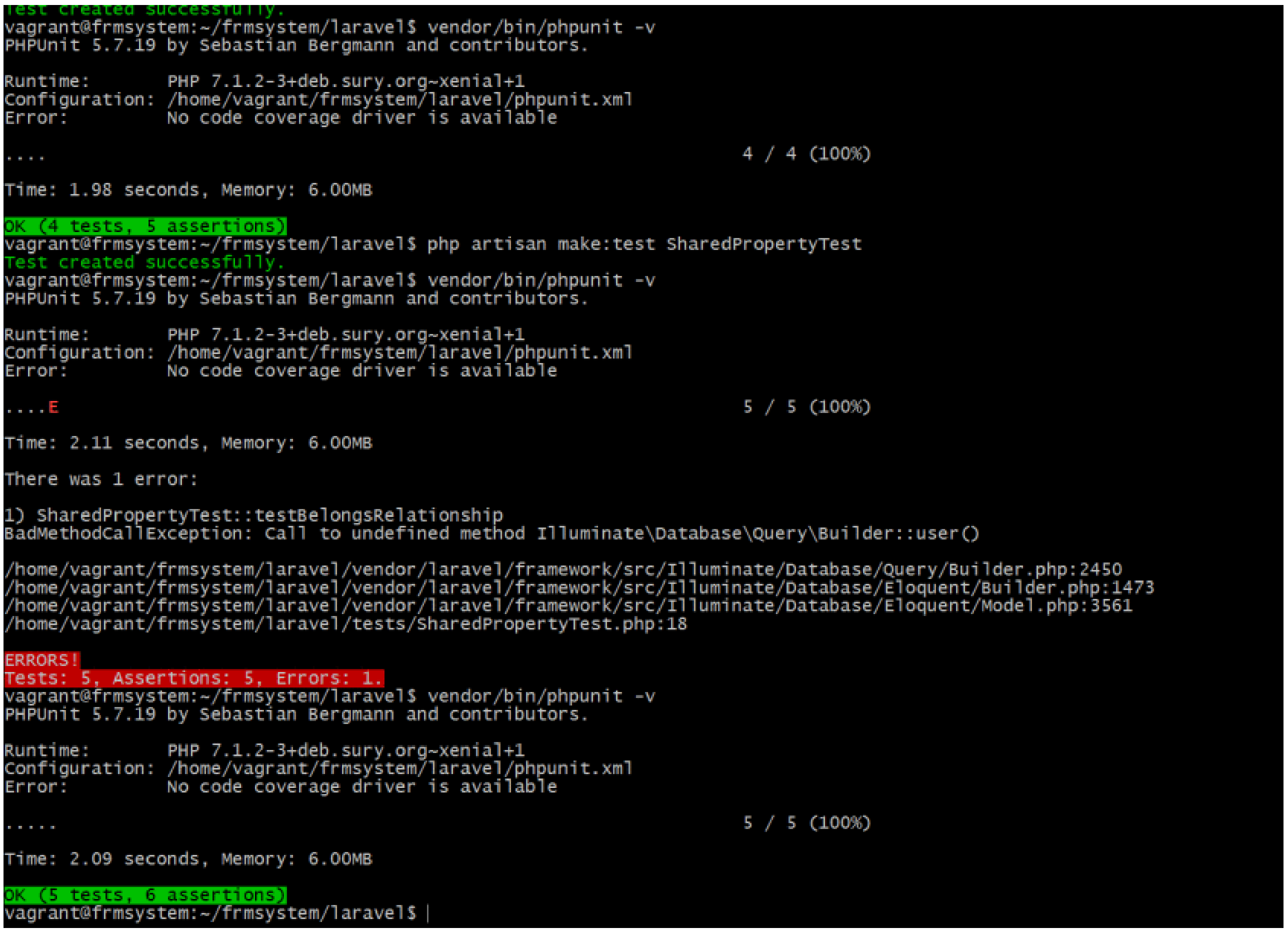
Hi guys,

in this blog post, I want to briefly talk you through our current setup for CI and testing. This is not final and may be subject to change in the next few weeks, since I am not 100% happy with the current state, but we have everything working as far as we need to.

First off we are using Travis CI to manage our testing. We have 2 branches that get tested whenever commits get pushed to it: One branch named “testing”. Whenever we have written tests for a feature branch, before merging into master, we use this branch to check that everything is working. If not, we can fix everything until we have a fully tested and working feature. We then know, that we can merge into master savely. This is also the second branch that gets tested by Travis CI, not only to reassure that everything is working 100%, but also because our (not yet) shiny badge in our Github Readme always resembles the status of the master branch, because the testing branch might get messed up in the process of development.

This badge is, by the way, part of Coveralls.io, that we use, to publish the code coverage information. It is hooked up into the chain after the tests, where the information gets uploaded to Coveralls by the Travis CI build. We have not written a lot of test code yet, I am just happy, that I was able to get the configuration running. The testing itself is done with PHPUnit, it was just the simplest for us to use and integrates well with Laravel. You can see that we are using it in our Composer configuration, where it can conveniently be installed.

The tests can easily be run inside of our Vagrant testing environment as can be seen in the following screenshot.



```
Test created successfully.
vagrant@frmsystem:~/frmsystem/laravel$ vendor/bin/phpunit -v
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.1.2-3+deb.sury.org~xenial+1
Configuration: /home/vagrant/frmsystem/laravel/phpunit.xml
Error:         No code coverage driver is available

....                                                  4 / 4 (100%)

Time: 1.98 seconds, Memory: 6.00MB

OK (4 tests, 5 assertions)
vagrant@frmsystem:~/frmsystem/laravel$ php artisan make:test SharedPropertyTest
Test created successfully.
vagrant@frmsystem:~/frmsystem/laravel$ vendor/bin/phpunit -v
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.1.2-3+deb.sury.org~xenial+1
Configuration: /home/vagrant/frmsystem/laravel/phpunit.xml
Error:         No code coverage driver is available

....E                                                 5 / 5 (100%)

Time: 2.11 seconds, Memory: 6.00MB

There was 1 error:

1) SharedPropertyTest::testBelongsRelationship
BadMethodCallException: Call to undefined method Illuminate\Database\Query\Builder::user()

/home/vagrant/frmsystem/laravel/vendor/laravel/framework/src/Illuminate/Database/Query/Builder.php:2450
/home/vagrant/frmsystem/laravel/vendor/laravel/framework/src/Illuminate/Database/Eloquent/Builder.php:1473
/home/vagrant/frmsystem/laravel/vendor/laravel/framework/src/Illuminate/Database/Eloquent/Model.php:3561
/home/vagrant/frmsystem/laravel/tests/SharedPropertyTest.php:18

ERRORS!
Tests: 5, Assertions: 5, Errors: 1.
vagrant@frmsystem:~/frmsystem/laravel$ vendor/bin/phpunit -v
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.1.2-3+deb.sury.org~xenial+1
Configuration: /home/vagrant/frmsystem/laravel/phpunit.xml
Error:         No code coverage driver is available

.....                                                 5 / 5 (100%)

Time: 2.09 seconds, Memory: 6.00MB

OK (5 tests, 6 assertions)
vagrant@frmsystem:~/frmsystem/laravel$ |
```

The data for the tests is generated by faker using Laravel's so called Model Factories. They are separate from the tests and if you have a look at the test code, you can see how we are only calling the factories for the tests to generate our testdata.

You can find the testplan in our github.

Also, we are now on Youtrack, where you can find our project here. We just had to switch, Jira is just too slow, we were not able to work off that anymore… That unfortunately also means, that all the data of the last semester will not be in our Gantt-chart at the end of the project. But this switch was really worth it, it seems that most things are far easier and clearer with Youtrack.

Best regards,

Daniel W.

24. April 2017 / 6 Comments

MOST RECENT POSTS

Search …

Follow •••