# Testing in Laravel

...

LaraHel Meetup 16.11.2016

Ville Glad ville.j.glad@gmail.com
Spend days at Isolta Oy

# This presentation is about

- Why test?
- Using PHPUnit is made easy
- Laravel integration package (former external package, since 5.1 included in Laravel)
- Integration vs Unit tests
- How to test simple things like registration or login

# Motivation to test

These are factors that motivates me to test:

- Save time
- Collaboration gets easier
- Most probably will make your code better
- Testing is fun
- Be more productive
  - May take more time initially, but take less time to develop further
- Not that fearful deployments

# Using PHPUnit on Laravel

- Most of the configs are already set
- What to do first
  - Add testing connection to config/database.php
    - ```
      'testing' => [
          'driver'   => 'sqlite',
          'database' => ':memory:',
          'prefix'   => '',
      ]
      ```
  - Add DB_CONNECTION environment setting to /phpunit.xml
    - `<env name="DB_CONNECTION" value="testing"/>`
- Now the tests use sqlite in-memory database to get db hitting tests work faster
  - Be aware of sqlite restrictions. databaseMigrations Trait migrates db before and rollback afterwards. So for example “Alter...” does not work.
      ```

# ExampleTest.php

- Extends abstract TestCase
  - Extends Illuminate\Foundation\Testing\TestCase
    - Extends PHPUnit_Framework_TestCase

# Test how your page should behave

Laravel provides nice features to test your MVC application. The simplest way to go from manual testing to automated is to write those manual steps into tests.

```
public function testLoginForm()
{
    $this->visit('/login')
         ->see('Login')
         ->type('user@example.com', 'email')
         ->type('mySecretToAccess', 'password')
         ->press('login')
         ->seePageIs('/admin');
}
```

https://laravel.com/api/5.3/Illuminate/Foundation/Testing/Concerns/InteractsWithPages.html

Pre-5.1 was this: https://github.com/laracasts/Integrated

# Integration vs Unit

- Integration
  - Single test hits more than one unit
  - Testing with database
  - Testing that designed functionality works
- Unit
  - Isolated testing of a function/method
  - Using test doubles to mock out the dependencies
  - Unit tests are way faster than integration ones

# Test your app is functioning like it is designed

- Start from user perspective (or API consumer)
  - Test the route and the controller are working
  - Test smaller units
    - Register-route and Registration controller method “register”
    - User-class has static create method for registering user
- Integration test for the whole functionality
  - Controller test
- Unit test individual components
  - Test model methods

# Isolated tests are the fastest

- Test only one unit at time
- Mock dependencies
- In Laravel:
  - Classes like models and services
  - Use what phpunit provides
  - Doubling tools:
    - Fake classes
    - Mockery
    - Prophecy (made for PHPSpec, included nowadays in PHPUnit)
    - $this->createMock()
  - https://phpunit.de/manual/current/en/test-doubles.html

# Examples: Testing flash

- Flash messages are stored on Session for only to the next request
- Asserting that session has the flash message
  - $this->assertSessionHas(‘flash_message’, [‘message’ => ...]);
  - $this->assertSessionMissing(‘flash_message’);

# Useful links

- https://laravel.com/docs/5.3/testing
- https://laravel.com/api/5.3/Illuminate/Foundation/Testing.html
- https://github.com/laravel/spark-tests/tree/3.0/tests
- https://adamwathan.me/2016/01/11/test-driven-laravel-from-scratch/
- https://phpunit.de/manual/current/en/index.html

# Conclusion

Laravel test helpers are a great asset for making testing more natural. Like coding in Laravel.

Beware of the tests slowing down if you have a larger application when using heavily integration tests.

Using database migrations or database transaction traits testing database is rolled back to its previous state.

Test is code as well. Need to be maintained.

Deploy more.