Tests

Tests: Introduction

A lot of people do not write tests, why?

Easy, writing tests takes time. You, your boss or you client may think it will make your development slower. Moreover, it increases your code base.

This is wrong, we will explain why.

Tests: Why?

Why you should write test

You will be more efficient
Manual testing takes time and effort, executing a test suite is more exhaustive and fast.

It shows other what you want to do
Nothing better than a good test to let others know how your code should behave.

Obviously, avoid regression
A test suite will make sure your code does not break something else in your project.

Reduce costs and win time
Eventually, having a nice test suite will reduce the time spent in bug fixing

It allows to control your code in the CI
Even if your code works in local environment, it could break later. Automated tests in the CI flow will prevent it.

Tests: Different types of tests

Unit tests

Most atomic tests, written to test one and only one feature or behaviour of a component.

Since your component may be connect to other parts of your code, and you only want to test that precise component, you may need to use mocks and/or fixtures.

- Mocks: a fake object / method that will simulate the behaviour of a real one. Commonly used to fake API calls. (ex: Nock for Node.js)
- Fixtures: an object that provides usable data. Can be random or fixed, based on what you want to test. (ex: Faker for PHP)

Tools

- PHP Unit
- PHP Spec
- Jest JS

Tests: Different types of tests

Integration tests

Often, your project will be connected to third parties or to external components.

Integration tests will ensure that these dependencies are still up and working as expected.

Tools

Depending on what you want to test, same tools as unit or functional tests

Tests: Different types of tests

Functional tests

Also known as end-to-end (E2E) tests, they focus on the behaviour of your application on the user point of view: how it will interact with your website, what behaviour should happen when browsing, clicking on a button, etc. Functional tests should also test your application under different conditions: differents browsers, network connections…

Tools

- Cypress
- Behat
- Selenium
- Even PHP Unit in some cases

Tests: What should I test?

EVERYTHING. Just kidding, you should **prioritize**.

1. Common and nominal cases
2. Edge cases, especially the ones that implies complex codes that probably have errors.
3. When you find a bug, fix it and write a test to cover it.
4. Everything else when possible.

What if I already have thousands of lines of code?

Same answer, prioritize. Begin to write code on your new implementations, take some time to cover the most common cases and the most buggy ones.

Tests: Practical work

- Clone and setup this project
- Fix the tests