Codeception Functional & Acceptance Tests

Sílvio Priem Mendes, PhD

Cofinanciado por:

POCH

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

- What if we could check our application without running it on a server?
- That way we could see detailed exceptions on errors, have our tests run faster, and check the database against predictable and expected results.

- That's what functional tests are for.

Functional Tests

- For functional tests, you emulate a web request ($_GET and $_POST variables) which returns the HTML response.
- Inside a test, you can make assertions about the response, and you can check if the data was successfully stored in the database.

- For functional tests, your application needs to be structured in order to run in a test environment.
- Codeception provides connectors to several popular PHP frameworks.

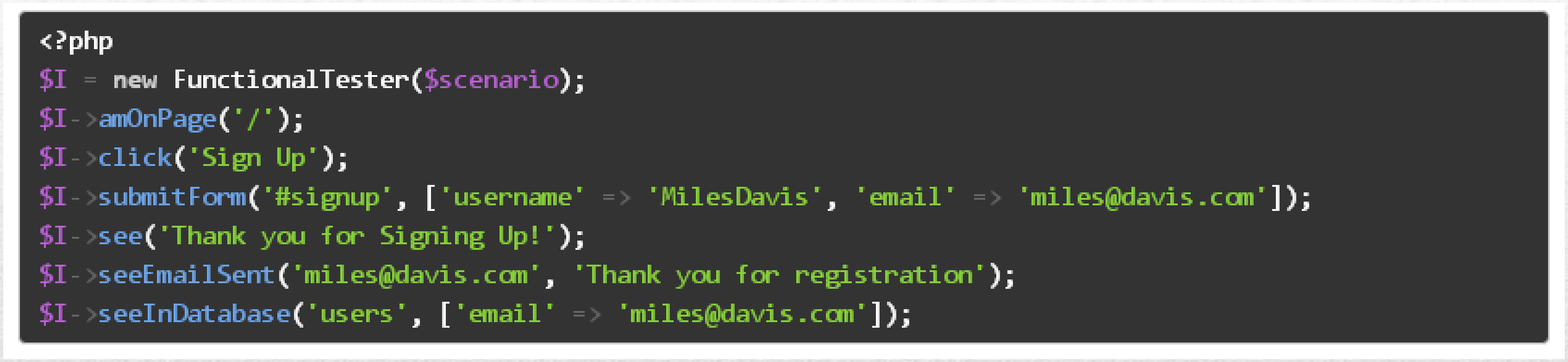
Functional Tests

- In simple terms we set the $_REQUEST, $_GET and $_POST variables and then we execute the application from a test.
- This may be valuable, as functional tests are faster and provide detailed stack traces on failures.
- Codeception can connect to different PHP frameworks that support functional testing: **Symfony2, Laravel5, Yii2, Zend Framework** and others. You just need to enable the desired module in your functional suite configuration to start.
- Modules for all of these frameworks share the same interface, and thus your tests are not bound to any one of them.

Functional Tests

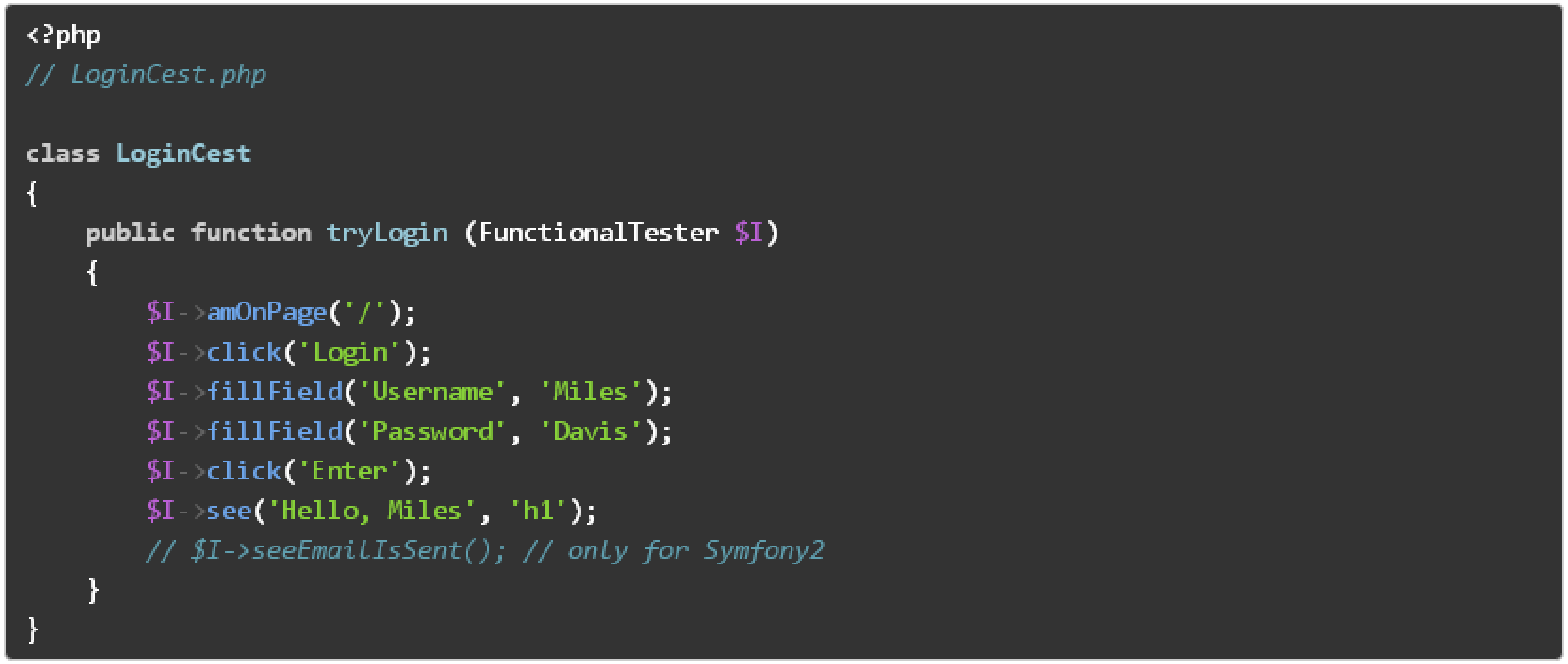
- Functional tests are great if you are using powerful frameworks.
- By using functional tests you can access and manipulate their internal state.
- This makes your tests shorter and faster.
- In other cases, if you don’t use frameworks there is no practical reason to write functional tests.

Functional Tests



```php
<?php
$I = new FunctionalTester($scenario);
$I->amOnPage('/');
$I->click('Sign Up');
$I->submitForm('#signup', ['username' => 'MilesDavis', 'email' => 'miles@davis.com']);
$I->see('Thank you for Signing Up!');
$I->seeEmailSent('miles@davis.com', 'Thank you for registration');
$I->seeInDatabase('users', ['email' => 'miles@davis.com']);
```

Functional Test Sample



```php
<?php
// LoginCest.php

class LoginCest
{
    public function tryLogin (FunctionalTester $I)
    {
        $I->amOnPage('/');
        $I->click('Login');
        $I->fillField('Username', 'Miles');
        $I->fillField('Password', 'Davis');
        $I->click('Enter');
        $I->see('Hello, Miles', 'h1');
        // $I->seeEmailIsSent(); // only for Symfony2
    }
}
```

- **the syntax is the same for functional and acceptance tests.**

Functional Test Sample 2

- Functional tests are usually much faster than acceptance tests.
- But functional tests are less stable as they run Codeception and the application in one environment.
- If your application was not designed to run in long lived processes (e.g. if you use the exit operator or global variables), then functional tests are feasible.

Functional Tests Limitations

Headers, Cookies, Sessions:

- One of the common issues with functional tests is the use of PHP functions that deal with headers, sessions and cookies.
- **The header function triggers an error if it is executed after PHP has already output something.**
- In functional tests we run the application multiple times, thus we will get lots of irrelevant errors in the result.

Functional Tests Limitations

External URL’s

- Functional tests cannot access external URL’s, just URL’s within your project.
- You can use Guzzle to open external URL’s (or any other library).

Functional Tests Limitations

Functional Tests Limitations

Shared Memory

- In functional testing, unlike running the application the traditional way, the PHP application does not stop after it has finished processing a request.
- Since all requests are run in one memory container, they are not isolated.
- If you see that your tests are mysteriously failing when they shouldn’t - try to execute a single test.
  - This will show if the tests were failing because they weren’t isolated during the run.
- Keep your memory clean, avoid memory leaks and clean global and static variables.

Enabling Functional Tests

Enabling Framework Modules

- You have a functional testing suite in the **tests/functional** directory.
- To start, you need to include one of the framework modules in the suite configuration file: **tests/functional.suite.yml**.

Yii2

- Yii2 tests are included in **Basic** and **Advanced** application templates.

- Codeception provides standard set of actions like `amOnPage`, `submitForm`, `see` for testing. Yii2 module provides special methods, like `amLoggedInAs` (for fast authentication),

YII2 Functional Tests

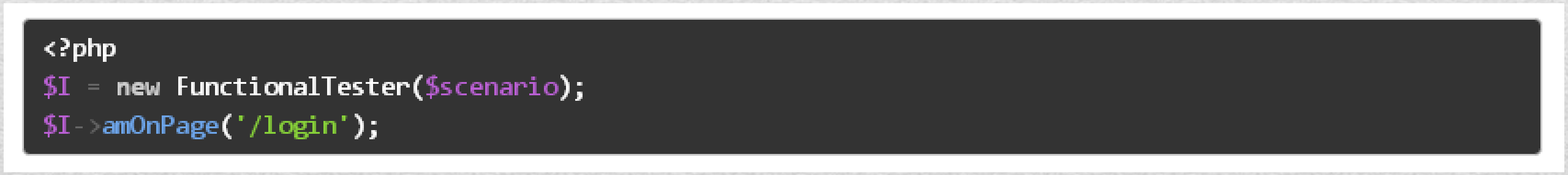
- Functional tests should be written inside **Cest** files, which is a scenario-driven test format of Codeception.
  - **Cest:** class-like structure for your tests

- You can easily create a new test by running:

```
./vendor/bin/codecept g:cest functional MyNewScenarioCest
```

Creating Functional Tests

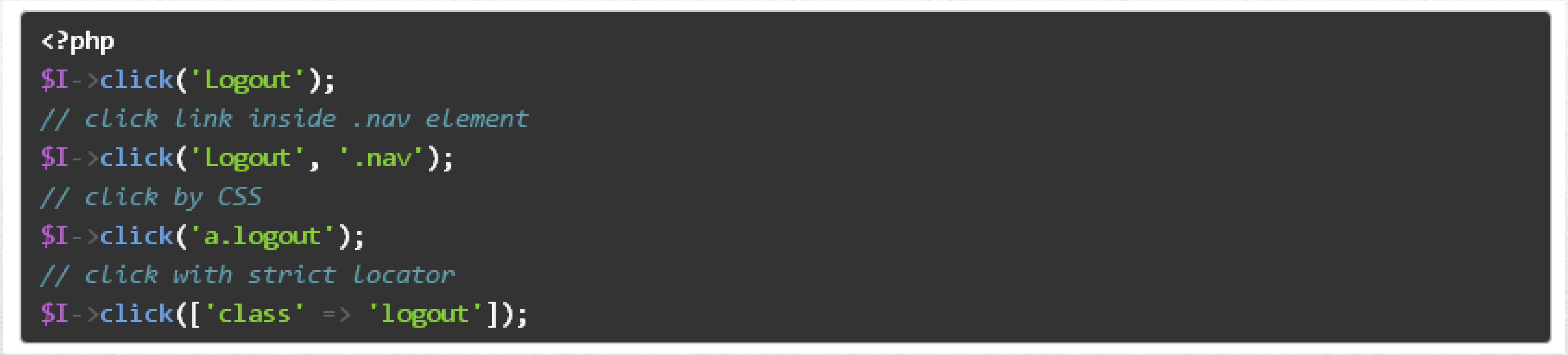
- Functional tests are written in the same manner as Acceptance Tests with the PhpBrowser module enabled.
- All framework modules and the PhpBrowser module share the same methods and the same engine.
- Therefore we can open a web page with amOnPage method:



```php
<?php
$I = new FunctionalTester($scenario);
$I->amOnPage('/login');
```

Writing Functional Tests

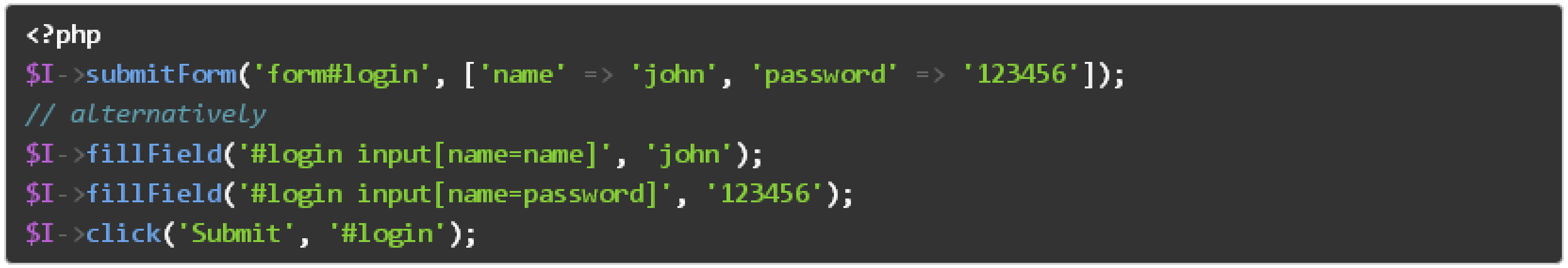
- We can click links to open web pages:



```php
<?php
$I->click('Logout');
// click link inside .nav element
$I->click('Logout', '.nav');
// click by CSS
$I->click('a.logout');
// click with strict locator
$I->click(['class' => 'logout']);
```

Writing Functional Tests

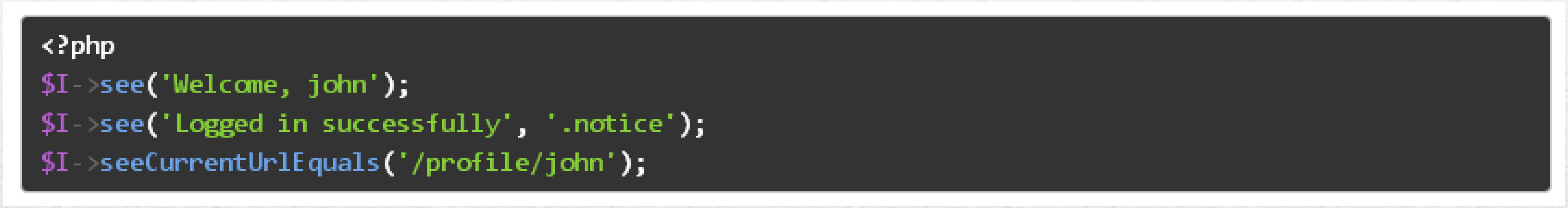
- We can submit forms as well:



```
<?php
$I->submitForm('form#login', ['name' => 'john', 'password' => '123456']);
// alternatively
$I->fillField('#login input[name=name]', 'john');
$I->fillField('#login input[name=password]', '123456');
$I->click('Submit', '#login');
```

Writing Functional Tests

- And do assertions:

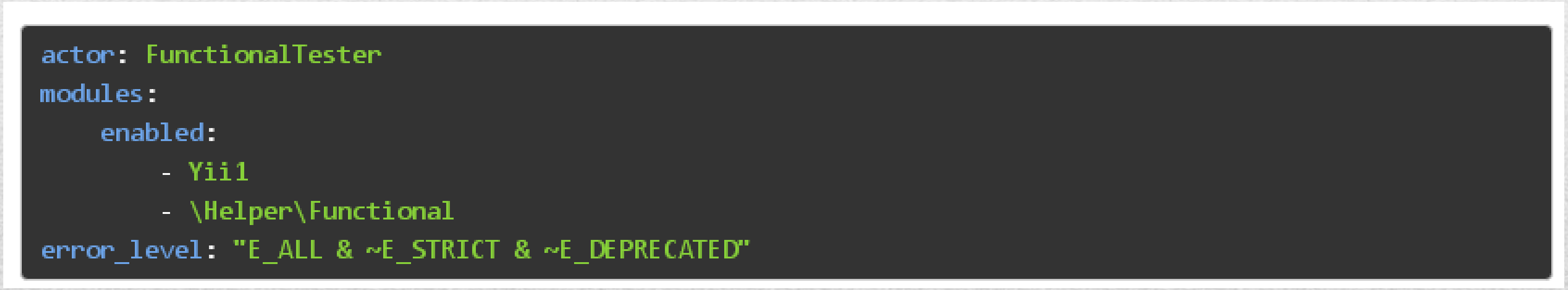


```php
<?php
$I->see('Welcome, john');
$I->see('Logged in successfully', '.notice');
$I->seeCurrentUrlEquals('/profile/john');
```

Writing Functional Tests

Error Reporting

- By default Codeception uses the **E_ALL & ~E_STRICT & ~E_DEPRECATED** error reporting level.
- In functional tests you might want to change this level depending on your framework's error policy.
- The error reporting level can be set in the suite configuration file:



```
actor: FunctionalTester
modules:
    enabled:
        - Yii1
        - \Helper\Functional
error_level: "E_ALL & ~E_STRICT & ~E_DEPRECATED"
```

Writing Functional Tests

- From a test perspective acceptance tests do the same as functional tests.
- They test the user interaction with application but in this case using real browser and web server (instead of simulating).
- They are much slower and much more fragile.
- API Methods are the same as in Functional Tests.
- They should not duplicate functional tests in matter of testing functionality but should be used for testing the UI of your application.

Acceptance Tests

- If you are unsure which tests should be acceptance and which are functional, write acceptance tests for JavaScript-rich applications, where UI highly depends on a browser processing.
- You can also use acceptance tests for happy-path scenarios, just to ensure that a real user using a real browser achieve the same results you expect in functional tests.

Acceptance Tests

- By default in basic application acceptance tests are disabled (as they require web server, Selenium Server and browser to be running). You can easily enable them by renaming **acceptance.suite.yml.example** to **acceptance.suite.yml**

```
mv tests/acceptance.suite.yml.example tests/acceptance.suite.yml
```

- Basic template uses codeception/base package which doesn't contain facebook/webdriver library required to run acceptance tests.

- Please change codeception/base to codeception/codeception in composer.json and run the update command.

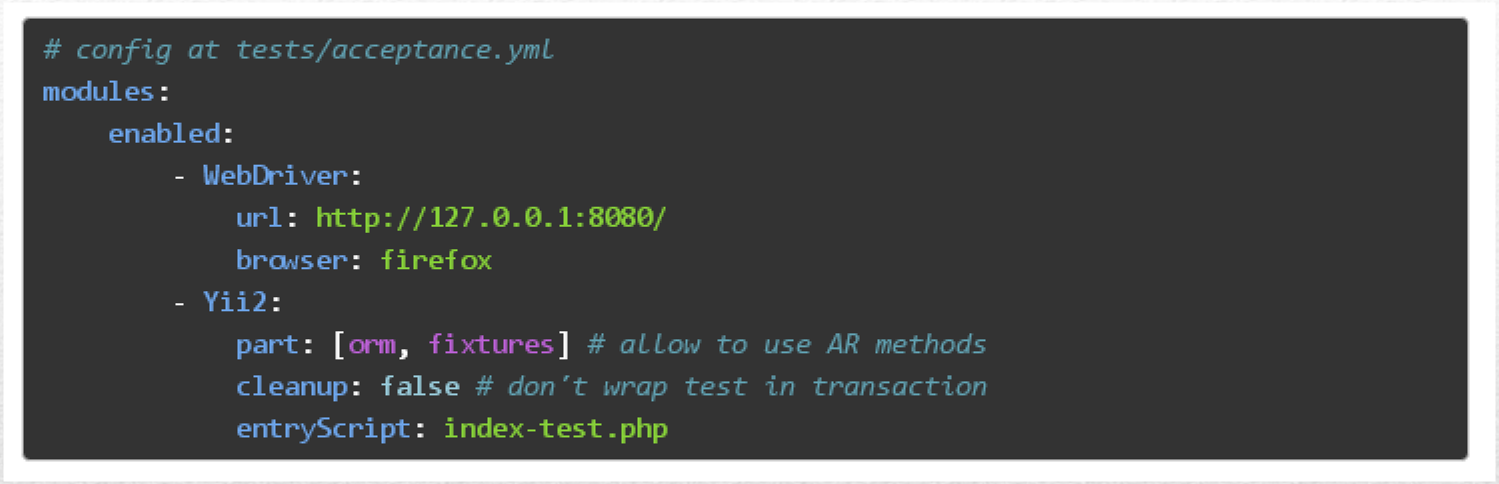
Acceptance Tests in YII

- Then you will need to launch application server in test mode:

```
./tests/bin/yii serve
```

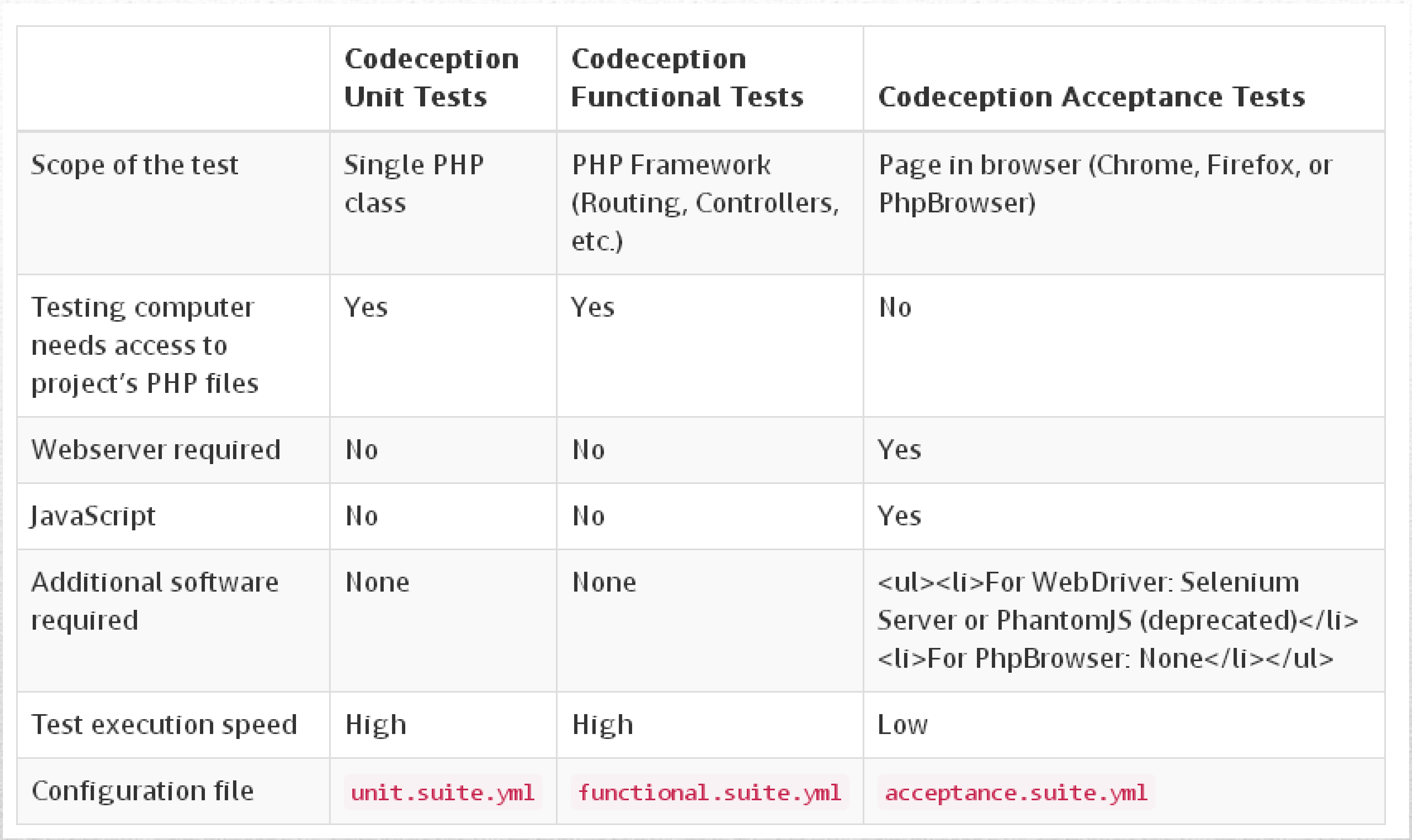
- and start a **Selenium Server**.
- The WebDriver module is used in acceptance tests.

Acceptance Tests in YII



```yaml
# config at tests/acceptance.yml
modules:
    enabled:
        - WebDriver:
            url: http://127.0.0.1:8080/
            browser: firefox
        - Yii2:
            part: [orm, fixtures] # allow to use AR methods
            cleanup: false # don't wrap test in transaction
            entryScript: index-test.php
```

Acceptance Tests in YII



	Codeception Unit Tests	Codeception Functional Tests	Codeception Acceptance Tests
Scope of the test	Single PHP class	PHP Framework (Routing, Controllers, etc.)	Page in browser (Chrome, Firefox, or PhpBrowser)
Testing computer needs access to project's PHP files	Yes	Yes	No
Webserver required	No	No	Yes
JavaScript	No	No	Yes
Additional software required	None	None	<ul><li>For WebDriver: Selenium Server or PhantomJS (deprecated)</li><li>For PhpBrowser: None</li></ul>
Test execution speed	High	High	Low
Configuration file	`unit.suite.yml`	`functional.suite.yml`	`acceptance.suite.yml`

Test Comparison Table