[Установка Codeception 2](#_Toc501702589)

[Первый вариант 2](#_Toc501702590)

[Глобальная установка. 2](#_Toc501702591)

[Локально: 2](#_Toc501702592)

[Второй вариант 3](#_Toc501702593)

[Третий вариант 4](#_Toc501702594)

[**Phar** 4](#_Toc501702595)

[Установка Selenium 4](#_Toc501702596)

[Первый вариант 4](#_Toc501702597)

[Второй вариант 4](#_Toc501702598)

[Третий вариант 4](#_Toc501702599)

[Инициализация окружения для тестирования Codecept 4](#_Toc501702600)

[Файловая структура 4](#_Toc501702601)

[Один загрузчик для нескольких приложений 6](#_Toc501702602)

[Пространства имен 6](#_Toc501702603)

[Bootstrap 7](#_Toc501702604)

[Настройки Codeception: 8](#_Toc501702605)

[codeception.yml 8](#_Toc501702606)

[functional.suite.yml 10](#_Toc501702607)

[acceptance.suite.yml 10](#_Toc501702608)

[Модули и помощники 11](#_Toc501702609)

[Стандартные модули 11](#_Toc501702610)

[Помощники 11](#_Toc501702611)

[Разрешение конфликтов 13](#_Toc501702612)

[Взаимодействие Модулей 13](#_Toc501702613)

[Неопределенные действия в помощниках 14](#_Toc501702614)

[Хуки 14](#_Toc501702615)

[Отладка 14](#_Toc501702616)

[Настройка 15](#_Toc501702617)

[Динамическая настройка 16](#_Toc501702618)

[МОДУЛЬ DB 16](#_Toc501702619)

[Dbh Module 19](#_Toc501702620)

[**Actions** 20](#_Toc501702621)

[Filesystem Module 20](#_Toc501702622)

[PhpBrowser Module 21](#_Toc501702623)

[Configuration 21](#_Toc501702624)

[Public Properties 21](#_Toc501702625)

[Actions 21](#_Toc501702626)

[Selenium Module 25](#_Toc501702627)

[Installation 25](#_Toc501702628)

[Configuration 25](#_Toc501702629)

[Public Properties 25](#_Toc501702630)

[Actions 25](#_Toc501702631)

[Unit Module 29](#_Toc501702632)

[Features 29](#_Toc501702633)

[Actions 30](#_Toc501702634)

# Установка Codeception

## Первый вариант

### Глобальная установка.

Если вы планируете использовать фреймворк Codeception для нескольких проектов, то лучше установить его глобально (не в каталог vendor текущего проекта, а в каталог vendor пользователя, откуда Composer так же подключает пакеты):

composer global require "codeception/codeception=2.0.\*"

composer global require "codeception/specify=\*"

composer global require "codeception/verify=\*"

Исполняемый файл **codecept.bat** при этом автоматически запишется в папку vendor\bin пользователя (для Win), которая должна быть указана в системной переменной **PATH** (как и папка с интерпретатором php) и таким образом вы сможете обращаться к фреймворку только по названию исполняемого файла:

codecept

Если в PATH не указан путь к **bin**, обязательно укажите. Чтобы проверить наберите в консоли:

echo %PATH%

Если планируете использовать кириллические символы (русский) в тестах, то чтобы они выводились корректно в консоли, можно добавить строку

@chcp 65001

в файл codecept.bat находящийся в папке пользователя, например: C:\Users\имя\_вашего\_пользователя\AppData\Roaming\Composer\vendor\bin\codecept.bat

При запуске команды

codecept

Мы должны увидеть приветствие:

Codeception version 2.0.16

### Локально:

Установка в текущий проект осуществляется командой:

composer require "codeception/codeception=2.0.\*"

composer require "codeception/specify=\*"

composer require "codeception/verify=\*"

Запускающий файл будет находиться в папке vendor\bin проекта. Чтобы не набирать для тестирования «vendor\bin\codecept»:  
**1) для Linux** создаем алиас:

alias codecept="./vendor/bin/codecept"

**2) для Win**  
создать в корне проекта файл codecept.bat с содержимым:

@echo off

@setlocal

@chcp 65001

set CODECEPT\_PATH=vendor/bin/

"%CODECEPT\_PATH%codecept.bat" %\*

@endlocal

строкой @chcp 65001 я указал кодировку чтобы кириллические символы отображались корректно.

При запуске команды

codecept

Мы должны увидеть приветствие:

Codeception version 2.0.16

Далее я буду описывать процесс исходя из того, что установлен Windows.  
Теперь для запуска из корня проекта достаточно писать просто

codecept

Если вы не планируете проводить приемочное тестирование, то можете установить базовую версию **Codeception** без загрузки библиотек Webdriver и Guzzle:

composer require codeception/base --dev

Для ознакомления с консольными командами **Codeception** можно воспользоваться справкой:

вывести описание доступных консольных команд

codecept

вывести справку по нужной команде

codecept название команды –h

## Второй вариант

Установка приложения локально для проекта

В файле composer.json добавляем

"require": {

"codeception/codeception": "2.0.\*",

}

И запускаем команду

composer update

## Третий вариант

## **Phar**

Скачайте phar-архив Codeception с http://codeception.com/thanks

wget http://codeception.com/codecept.phar .

# Установка Selenium

## Первый вариант

Качаем файл <https://cdn.rawgit.com/ilopX/web-demos/master/demos/console/Codeception-install/run-selenium-server.php>

## Второй вариант

<http://www.seleniumhq.org/download>

<http://chromedriver.storage.googleapis.com/index.html>

Создать и запустить start-server.bat

java -jar c:\selenimu-server\selenium-server-standalone{version}.jar -Dwebdriver.chrome.driver=c:\selenimu-server\chromedriver.exe

## Третий вариант

Но чтобы не возиться со скачиванием файла вручную и созданием ярлыка для его запуска лучше просто установим сам сервер и запускающий скрипт через Composer:

composer global require se/selenium-server-standalone

и, если у вас путь к bin уже прописан в PATH, будем запускать в консоли просто по этому же имени:

selenium-server-standalone

#### SELENIUM.

**PhpBrowser** не позволяет полноценно протестировать страницу на которой используется **JavaScript**, т.к. данный модуль использует библиотеку Curl. Для таких целей подключают модуль WebDriver, который использует **Selenium**.   
Selenium — это проект, предоставляющий драйвера для разных браузеров, которые встраиваются в них и позволяют управлять ими. Также, Selenium содержит сервер, который позволяет управлять большим числом разных браузеров и распределять задания между ними. Selenium сервер написан на Яве, потому она понадобится чтобы его запустить.  
  
При использовании Selenium для тестов будет запускаться браузер и вы увидите, как происходит заполнение полей форм, их отправка на сервер, клики по элементам и прочие действия, которые обычно совершаются пользователем приложения.  
  
В примерах я буду использовать браузер **Firefox**, поэтому в начале указываю как настроить их взаимодействие с Selenium. Возможно так же использование браузера Google Chrome, пример настройки в конце статьи.  
  
Есть несколько способов установки Selenium.

##### 1. Самый простой и удобный с помощью Composer.

Страница расширения <https://github.com/sveneisenschmidt/selenium-server-standalone>   
Выполнив команду:

composer global require --dev se/selenium-server-standalone

вы установите сервер **Selenium** глобально в папку текущего пользователя, у меня это C:\Users\Сергей\AppData\Roaming\Composer\vendor  
  
Или с указанием конкретной версии:

composer global require --dev se/selenium-server-standalone 2.53.1

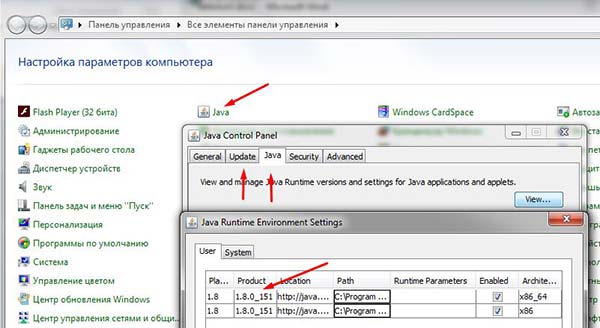
Тут я указал версию 2.53.1, которая будет работать с версией **Java**, установленной сейчас на одном из моих компьютеров (обновить ее я не могу). Можно устанавливать без указания версии, в таком случае установится новейшая версия.   
  
Стоит отметить что для использования данной версии Seleniumа (2.53.1), браузер Firefox должен быть ниже версии 48, например 47.0.1  
  
Если вы устанавливаете новую версию **Seleniumа** (>=3), то потребуется и новая версия Firefox и что еще важно потребуется дополнительно загрузить **geckodriver**. Подробнее про него тут: [http://barancev.github.io/geckodriver](http://barancev.github.io/geckodriver/)  
Скачиваем данный драйвер <https://github.com/mozilla/geckodriver/releases> для своей операционной системы и сохраняем его в каталог указанный в системной переменной **PATH**. Я сохранил в D:\OpenServer 5.2.7\modules т.к. эта папка указана в списке. Или указываем нужный путь к каталогу с драйвером в переменной системной переменной PATH.  
  
  
Запуск сервера Selenium осуществляется командой в консоли:

selenium-server-standalone

Если вы запускаете из Windows и у вас возникает такая ошибка:  
/usr/bin/bash: "C:\Users\...\AppData\Roaming\Composer\vendor\bin\/../se/selenium-server-standalone/bin/selenium-server-standalone": No such file or directory  
то нужно подкорректировать файл C:\Users\{USERNAME}\AppData\Roaming\Composer\vendor\bin\selenium-server-standalone.bat (тут показан путь при глобальной установке Selenium), заменить 2 последние строки на

SET BIN\_TARGET=%~dp0/../se/selenium-server-standalone/bin/selenium-server-standalone.jar

java -jar "%BIN\_TARGET%" %\*

т.к. прописанная там по-умолчанию команда **sh** предназначена для выполнения на **Unix** подобных системах.  
  
А если при выполнении команды возникнет ошибка такого плана:  
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/openqa/grid/selenium/GridLauncherV3 : Unsupported major.minor version 52.0…  
значит **Selenium** не может работать с установленной версией **Java** и нужно ее обновить или установить другую версию Selenium. В **Win-7** для просмотра версии или обновления Java можно открыть в «Панели управления»:   
  
  
Если все равно будут ошибки при запуске Selenium установите версию пониже.  
  
Так же, при экспериментировании с версиями, уже при запуске тестов возможна ошибка такого плана:   
[Facebook\WebDriver\Exception\UnknownServerException] Unable to connect to host 127.0.0.1 on port 7055 after 45000 ms.…  
Для устранения данной ошибки нужно обновить (или подобрать) версию Firefox с которой работает скаченный модуль Selenium. Например с selenium-server-standalone-2.53.1.jar будет работать с Firefox 47.0.1

##### 2. Скачиваем и запускаем Selenium непосредственно.

Переходим на http://www.seleniumhq.org/download/ и нажимаем по ссылке после слов «Download version».  
Для того чтобы скачать предыдущие версии Selenium переходим по ссылке http://selenium-release.storage.googleapis.com/index.html и там скачиваем файл такого типа: selenium-server-standalone-X.X.X.jar  
  
Далее, для использования **Firefox** скачиваем драйвер <https://github.com/mozilla/geckodriver/releases> для своей операционной системы и сохраняем его в каталог указанный в системной переменной **PATH**. Я сохранил в D:\OpenServer 5.2.7\modules, эта папка указана у меня в PATH. Вы можете указать нужный путь к каталогу с драйвером в переменной PATH. В эту же папку я переместил и скачанный модуль Selenium.  
  
Установка описана в комментариях файла модуля - vendor\codeception\codeception\src\Codeception\Module\WebDriver.php  
  
Перед выполнением тестирования должен быть запущен сервер. Запускаем командой:

java -jar -Dwebdriver.gecko.driver="D:\OpenServer 5.2.7\modules\geckodriver.exe" "D:\OpenServer 5.2.7\modules\selenium-server-standalone-3.4.0.jar"

где нужно указать свои пути к скачанным файлам и в конце изменить версию **Selenium**.  
Тут я указал путь к **geckodriver.exe**, но если он у вас лежит в папке которая указана в системной переменной PATH, то его можно не указывать. Получится:

java -jar "D:\OpenServer 5.2.7\modules\selenium-server-standalone-3.4.0.jar"

Так же вы можете создать файл selenium-server-standalone.bat в папке vendor\bin с указание своих путей к исполняемым файлам для удобства запуска модуля одной короткой строкой по примеру установки с помощью Composer.  
  
Если возникнет ошибка примерно с таким содержанием:  
[Facebook\WebDriver\Exception\SessionNotCreatedException] Unable to create new remote session. desired capabilities = Capabilities [{firefoxOptions=org.openqa.selenium.firefox.FirefoxOptions@7a99c504, browserName=firefox, moz:firefoxOptions=org.openqa.selenium.firefox.FirefoxOptions@7a99c504}], required capabilities = Capabilities [{}]Build info: version: '3.0.0', revision: '350cf60', time: '2016-10-13 10:48:16 -0700'System info: host: 'MICROSOFT-PC', ip: '192.168.56.1', os.name: 'Windows 7', os.arch: 'amd64', os.version: '6.1', java.version: '1.8.0\_40'Driver info: driver.version: FirefoxDriver  
значит запускаемый файл Selenium не соответствует установленной версии Java и скорее всего придется попробовать Selenium более старой версии.  
  
См. так же ошибки описанные в пункте установки с помощью Composer.  
  
У меня работала такая связка:  
Java 1.8.0\_40 (1.8.0\_51)selenium-server-standalone-3.4.0.jarи Firefox 56.0.2  
хотя текущая на момент написания статьи была версия selenium-server-standalone-3.6.0.jar

##### Настройка модуля **WebDriver** (Selenium).

В файле tests/acceptance.suite.yml

actor: AcceptanceTester

modules:

enabled:

- WebDriver:

url: 'http://site.loc/'

browser: 'firefox'

тут:  
- включаем модуль **WebDriver**;  
- в “url” указать стартовый URL вашего сайта;  
- в browser указываем браузер с которым будем работать 'chrome' или 'firefox'. Мы пишем 'firefox'.  
  
**Возможные настройки модуля WebDriver**:

* **url** - Стартовый URL для вашего приложения.
* **browser** - Браузер для запуска.
* **host** - хост сервера Selenium (по умолчанию 127.0.0.1).
* **port** - порт сервера Selenium (по умолчанию 4444).
* **restart** - установите в «false» (по умолчанию) для использования одного и того же окна браузера для всех тестов или установите «true» для создания нового окна для каждого теста. В любом случае, когда все тесты завершены, окно браузера закрывается.
* **start** - автозапуск браузера для тестирования. Может быть отключен, если сеанс браузера начинается с «\_initializeSession» внутри помощника.
* **window\_size** - Начальный размер окна. Установите значение «maximize» или пропишите в формате «640x480».
* **clear\_cookies** - Установите значение «false», чтобы сохранить файлы cookie, или установите значение «true» (по умолчанию), чтобы удалить все куки между тестами.
* **wait** (по умолчанию: 0 секунд) - всякий раз, когда элемент требуется и не находится на странице, подождите несколько секунд, чтобы найти его перед сбоем.
* **capability** - Настраивает Selenium [желаемые возможности] (https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities). Должен быть массив значений ключа.
* **connection\_timeout** - тайм-аут для открытия соединения с удаленным сервером (по умолчанию 30 секунд).
* **request\_timeout** - тайм-аут для запроса вернуть что-то из удаленного сервера selenium (по умолчанию 30 секунд).
* **pageload\_timeout** - время ожидания загрузки страницы до того, как будет выдана ошибка (по умолчанию 0 секунд).
* **http\_proxy** - устанавливает URL-адрес прокси-сервера HTTP для тестирования удаленного сервера.
* **http\_proxy\_port** - устанавливает порт HTTP-прокси-сервера
* **debug\_log\_entries** - сколько записей Selenium для печати с «debugWebDriverLogs» или при сбое (по умолчанию 15).
* **log\_js\_errors** - установите значение «true», чтобы включить возможный отчет JavaScript в HTML, или установите значение «false» (по умолчанию) для деактивации.

Подробнее можно посмотреть в блоке ## Configuration файла vendor\codeception\codeception\src\Codeception\Module\WebDriver.php  
  
При использовании Selenium, метод **see()** позволяет не просто убедится, что текст внутри элемента существует, но также проверит, что элемент действительно виден пользователю.

<?php

$I->see('Confirm','#modal');

?>

Например на странице используется «Bootstrap Accordion» т.е. выводятся в ряд заголовки, кликнув по которым открывается блок соответствующего им текста. До клика по нужному заголовку, блок принадлежащего ему текста скрыт. При использовании **PhpBrowser** проверка текста из скрытого блока на то, что он в данное время не видим пользователю:

$I->dontSee('скрытый текст');

завершится с ошибкой, т.к. PhpBrowser не умеет работать с скрытым текстом и он будет считаться видимым. А вот **Selenium** с этим справится.

# Инициализация окружения для тестирования Codecept

Сodecept bootstrap

эта команда сгенерирует папку с тестами и установит дерево нужных каталогов с файлами.

Команда инициализации создаст следующую структуру файлов и каталогов в указанной вами директории.

### Файловая структура

app/tests/codeception Корневой каталог фреймворка

├── codeception.yml Основной файл конфигурации

└── tests Каталог с тестами

├── \_bootstrap.php Общий файл инициализации

├── unit.suite.yml Конфигурация Unit-тестов

├── unit Каталог с Unit-тестами

│ ├── \_bootstrap.php Файл инициализации при запуске Unit-тестов

│ └── CodeEngineer.php Базовый класс с общими методами необходимыми в Unit-тестах

├── functional.suite.yml Конфигурация модулей функциональных тестов

├── functional Каталог с функциональными тестами

│ ├── \_bootstrap.php

│ └── TestEngineer.php

├── acceptance.suite.yml Конфигурация модулей приемочных тестов

├── acceptance Каталог с приемочными тестами

│ ├── \_bootstrap.php Файл инициализации при запуске приемочных тестов

│ ├── WebEngineer.php Базовый класс с кастомными тестовыми методами

│ └── ManageUserCest.php Класс тестирует интерфейс управления пользователями

├── \_data Каталог для файлов, необходимых при тестировании

│ ├── database.dump.sql Дамп БД для восстановления тестового окружения приложения

│ ├── books.csv Вы можете держать здесь любые необходимые длятестов файлы..

│ └── testing.png Например, изображения...

├── \_pages Каталог для файлов описанных данных для заполнения страниц

│ └── AboutPage.php Данные для заполнения на странице About

│ └── contactPage.php Данные для заполнения на странице Contact

├── \_helpers

│ ├── CodeHelper.php

│ ├── TestHelper.php

│ └── WebHelper.php

└── \_log Логи, отчеты, скриншоты о прохождении тестов

├── report.json

├── report.tap.log

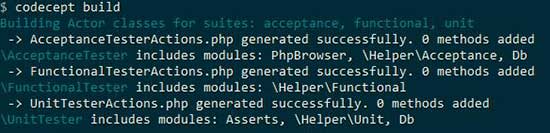
└── ManageUserCest.create.fail.png

Основной концепцией Codeception является представление тестов как действий человека. Мы будем называть этого человека "**парнем**" (Guy). У нас есть **CodeGuy**, который выполняет функции/методы и тестирует код. Также у нас есть **TestGuy** - опытный тестировщик, который тестирует приложение целиком и знает кое-что о его внутреннем устройстве. Еще у нас есть **WebGuy**, пользователь, который работает с нашим приложением через интерфейс, который мы предоставляем.

Каждый из этих "парней" представляет из себя PHP класс с набором действий, который он может совершать. Как Каждый из этих "парней" имеет различные способности. Эти способности не постоянны, Вы можете расширять их. Вы даже можете создавать новых "парней", но запомните: один "парень" на один набор тестов.

Классы "парней" не пишутся, они генерируются следующей командой после изменения конфигурационных файлов:

codecept build

это сгенерирует трейты с методами для тестирования из модулей Codeception которые появится в папке tests\\_support\\_generated.   
При этом в информации, которая будет выведена в консоль после выполнения команды, можно увидеть кол-во добавленных методов и подключенных модулей для каждого типа теста:  
  
  
Сам класс конфигурации находится в файле vendor\codeception\codeception\src\Codeception\Configuration.php

## Один загрузчик для нескольких приложений

В случае, если проект состоит из нескольких приложений (frontend, admin, api) или вы используете бандлы Symfony2,  
вы можете захотеть, чтобы тесты для всех приложений запускались одним загрузчиком.  
В таком случае вы получите один отчет для всего проекта.

Начиная с версии Codeception 1.6.3 появилась возможность создать meta-config, включающий конфигурационные файлы из разных каталогов.

Поместите codeception.yml в корень проекта, и укажите пути к других codeception.yml конфигам, которые вы хотите подключить.

include:

- frontend

- admin

- api/rest

paths:

log: log

settings:

colors: false

Кроме того, вам следует указать путь к каталогу log, в котором будут храниться отчеты и логи всего проекта.

# Пространства имен

Для того, чтобы избежать конфликтов имен между Guy и Helper классами, они должны быть добавлены в пространства имен.  
Для создания набора тестов с использованием пространств имен, добавьте опцию --namespace к команде bootstrap.

codecept bootstrap --namespace frontend

Это создаст новый проект с параметром namespace: frontend в файле конфигурации codeception.yml.  
Помощники будут использовать пространство имен frontend\Codeception\Module а классы Guy пространство имен frontend.  
Сгенерированные классы будут выглядеть примерно так:

<?php use frontend\WebGuy;

$I = new WebGuy($scenario);

//...

?>

Codeception имеет утилиты для того, чтобы обновить тесты существующего проекта, добавив в них использование пространств имен. Этого можно добиться выполнив следующую команду

codecept refactor:add-namespace frontend

Вы получите guy классы и помощники, а так же cept тесы обновленные для использования с пространствами имен. Запомните, что Cest файлы должны быть обновлены вручную. Коме того опция namespace не изменит пространство имен тестов или Cest классов. Она используется только для Guys классов или классов Помощников.

В то время, когда каждое приложение (бандл) имеет собственное пространство имен и разные классы помощники и guy классы, вы можете выполнить их с помощью одной команды.  
Используйте meta-config который мы создали выше для запуска тестов как вы делаете это обычно.

codecept run

Это выполнит тесты для всех 3 приложений и сольет отчеты в один. По существу, это может быть довольно удобно, в случае, если вы запускаете тесты на сервере непрерывной интеграции и хотите получить один отчет в формате JUnit или HTML. Отчет о покрытии кода тестами, будет так же объединен в один.

## Bootstrap

Каждый набор тестов имеет свой собственный стартовый (bootstrap) файл. Он расположен в директории, содержащей набор тестов и называется \_bootstrap.php. Этот файл исполняется перед каждым тестом из соответствующего набора. Любые подготовительные операции для набора тестов следует писать именно в этом файле

В папке tests в файле \_bootstrap.php необходимо написать конфигурацию общую для 3 типов тестов.

В файле codeception/\_bootstrap.php переопределяем путь к базе данных. Для тестов мы можем использовать SQL

К примеру

<?php  
ini\_set('xdebug.max\_nesting\_level', 1000);

defined('YII\_DEBUG') **or** define('YII\_DEBUG', **true**);  
defined('YII\_ENV') **or** define('YII\_ENV', 'test');  
  
**require**(*\_\_DIR\_\_* . '/../vendor/autoload.php');  
**require**(*\_\_DIR\_\_* . '/../vendor/yiisoft/yii2/Yii.php');  
  
Yii::*setAlias*('@tests', dirname(*\_\_DIR\_\_*) . '/tests');

Изначально нас интересуют эти файлы:

tests

└── codeception

├── acceptance

│ └── \_bootstrap.php

├── functional

│ └── \_bootstrap.php

└── unit

└── \_bootstrap.php

Файлы acceptance/\_bootstrap.php

**new** yii\web\Application(**require**(dirname(\_\_DIR\_\_) . '/config/acceptance.php'));

и:

functional/\_bootstrap.php

**new** yii\web\Application(**require**(dirname(\_\_DIR\_\_) . '/config/functional.php'));

Внутри же третьего unit/\_bootstrap.php такого кода нет, так как создание и завершение приложения находятся в методах setUp() и tearDown() самого класса yii\codeception\TestCase, от которого будут наследоваться классы наших тестов из папки unit.

# Настройки Codeception:

Для каждого типа тестов (приемочные, функциональные, модульные) существует дополнительный, специализированный конфигурационный файл.

Codeception имеет глобальный файл конфигурации codeception.yml и отдельный конфигурационный файл для каждого набора тестов. Также поддерживаются конфигурационные файлы .dist. Если у Вас в проекте несколько разработчиков - разместите общие параметры конфигрурации в файле codeception.dist.yml, а индивидуальные настройки каждого разработчика в файле codeception.yml. Тоже самое справедливо и для конфигураций наборов тестов. Например, файл unit.suite.yml будет объединен с файлом unit.suite.dist.yml.

tests

├── codeception

│ ├── acceptance.suite.yml

│ └── acceptance.suite.dist.yml

│ ├── functional.suite.yml

│ └── functional.suite.dist.yml

│ ├── unit.suite.yml

│ └── unit.suite.dist.yml

│ ├ codeception.yml

└codeception.dist.yml

### codeception.yml

В общем файле codeception.yml настроим анализатор покрытия (test coverage) нашего кода. Включаем только папки с кодом и исключаем всё лишнее. Другие секции нас вполне устраивают, и мы будем запускать тесты по адресу http://localhost:8080, поэтому всё остальное оставляем как есть:

actor: Tester

coverage:

enabled: true

whitelist:

include:

- ../components/\*

- ../mail/\*

- ../modules/\*

- ../views/\*

exclude:

- ../modules/admin/messages/\*

- ../modules/main/messages/\*

- ../modules/user/messages/\*

blacklist:

include:

- ../assets/\*

- ../config/\*

- ../runtime/\*

- ../environments/\*

- ../messages/\*

- ../vendor/\*

- ../web/\*

- ../tests/\*

paths:

tests: codeception

log: codeception/\_output

data: codeception/\_data

helpers: codeception/\_support

settings:

bootstrap: \_bootstrap.php

suite\_class: \PHPUnit\_Framework\_TestSuite

memory\_limit: 1024M

log: true

colors: true

config:

# the entry script URL (with host info) for functional and acceptance tests

# PLEASE ADJUST IT TO THE ACTUAL ENTRY SCRIPT URL

test\_entry\_url: http://localhost:8080/index-test.php

Оставляем его тоже.

По умолчанию главный файл конфигурации codeception.yml выглядит приблизительно вот так:

actor: Engineer

paths:

# where the modules stored

tests: tests

# logs and debug

# outputs will be written there

log: tests/\_log

# directory for fixture data

data: tests/\_data

# directory for custom modules (helpers)

helpers: tests/\_helpers

settings:

# name of bootstrap that will be used

# each bootstrap file should be

# inside a suite directory.

bootstrap: \_bootstrap.php

# You can extend the suite class if you need to.

suite\_class: \PHPUnit\_Framework\_TestSuite

# by default it's false on Windows

# use [ANSICON](http://adoxa.110mb.com/ansicon/) to colorize output.

colors: true

# Tests (especially functional) can take a lot of memory

# We set a high limit for them by default.

memory\_limit: 1024M

# If a log should be written.

# Every action in test is logged.

# Logs are kept for 3 days.

log: true

# Global modules configuration.

modules:

config:

Db:

dsn: 'pgsql:host=localhost;dbname=DBNAME'

user: 'USER'

password: 'PASS'

dump: tests/\_data/database.dump.sql

populate: true

cleanup: true

### functional.suite.yml

Теперь подключим модуль фикстур для наших функциональных и приёмочных тестов. Откроем functional.suite.yml и добавим его в массив активных модулей:

class\_name: FunctionalTester

modules:

enabled:

- Filesystem

- Yii2

- tests\codeception\\_support\FixtureHelper

config:

Yii2:

configFile: 'codeception/config/functional.php'

### acceptance.suite.yml

Аналогично добавим модуль фикстур в acceptance.suite.yml. Заодно и вместо curl-эмулятора браузера PhpBrowser подключим Facebook WebDriver, предназначенный для работы с Selenium Server, и настроим его на запуск браузера Firefox:

class\_name: AcceptanceTester

modules:

enabled:

- WebDriver

- tests\codeception\\_support\FixtureHelper

config:

WebDriver:

url: http://localhost:8080

browser: firefox

restart: true

window\_size: 1024x768

На этом конфигурирование завершено.

##### Быстрая очистка таблицы.

Использование способа очистки таблицы перезаписью дампа база данных довольно долгий способ. Можно воспользоваться методом **load()** класса Codeception\Lib\Driver\Db который позволяет выполнить произвольный запрос к БД.  
В файле нужного помощника (tests\\_support\Helper):

public function clearTable($table){

$db = $this->getModule('Db')->driver;

$db->load(["TRUNCATE TABLE `$table`"]);

}

В нужном тесте:

$I->clearTable('название таблицы');

эту строку можно разместить как только в нужном методе так и в методах **\_before()** или **\_after()**.

Если вы создадите в каталоге проекта файл **.scrutinizer.yml и поместите туда вот такое содержание**

imports:

- php

tools:

external\_code\_coverage:

timeout: 2100 # Timeout in seconds.

# disable copy paste detector and similarity analyzer as they have no real value

# and a huge bunch of false-positives

php\_sim: false

php\_cpd: false

**то ваш проект на GitHub будет тестировать сервис scrutinizer-ci.com**

# Модули и помощники

Codeception использует модульную архитектуру чтобы создать комфортное тестовое окружение для каждого написанного вами набора тестов.  
Модули позволяют выбрать действия (actions) и утверждения (assertions), которые будут выполнены в тестах.

Все действия и утверждения, которые будут выполнены "объектом-парнем" объявлены в модулях. Это может выглядеть будто Codeception ограничивает вас в тестировании, но это не правда. Вы можете расширить набор тестов своими собственными действиями и утверждениями, описав их в своем модуле.

Давайте посмотрим на этот тест.

<?php

$I = new TestGuy($scenario);

$I->amOnPage('/');

$I->see('Hello');

$I->seeInDatabase('users', array('id' => 1));

$I->seeFileFound('running.lock');

?>

Тест может оперировать различными сущностями: страница может быть загружена с помощью модуля Symfony1, база данных проверена с помощью модуля Db, а наличие файла может быть с помощью модуля Filesystem.

Модули привязан к классам "Парней" c помощью файла настроек тестового набора.  
Например, в tests/functional.suite.yml мы можем увидеть.

class\_name: TestGuy

modules:

enabled: [Symfony1, Db, Filesystem]

У класса TestGuy есть методы объявленные в модулях. На самом деле, класс не содержит все эти методы, но он выступает в качестве прокси для них. Он знает какой модуль выполняет данное действие и передает ему параметры. Чтобы ваша IDE могла видеть все методы класса TestGuy используйте команду build. Команда генерирует определение класса TestGuy, копируя сигнатуры методов из настроенных модулей.

## Стандартные модули

## https://github.com/Codeception/Codeception/tree/master/docs/modules

## Помощники

Codeception не ограничивает вас в использовании модулей только из основного репозитория. Несомненно, в вашем проекте вы можете захотеть добавить собственные действия в тестовые наборы. Запустив команду bootstrap, Codeception сгенерирует для вас три заготовленных модуля, по одному для каждого набора. Эти модули называются 'Помощниками', их можно найти по следующему пути tests/helpers.

Это хорошая идея определять недостающие действия или команды утверждения (assertion) в помощниках.

Скажем, мы собираемся расширить класс TestHelper. По умолчанию он связан с классом TestGuy и набором функциональных тестов.

Что касается действий, все очень просто. Каждое действие, которое вы определяете является публичной функцией. Опишите публичную функцию, запустите команду build и вы увидите, что добавлена новая функция в класс TestGuy. Примечание: публичные методы с префиксом \_ рассматриваются как скрытые и не будут добавлены в класс вашего "Парня".

С утверждениями немного сложнее. В первую очередь, рекомендуется добавлять префикс see или donSee для всех утверждающих действий. В философии Codeception, все тесты выполняются людьми, например "парнями". Ожидаемый результат, который они видят (или не видят) это то, что мы используем для утверждения.

Называйте ваши утверждения подобным образом:

<?php

seePageReloaded();

seeClassIsLoaded($classname);

dontSeeUserExist($user);

?>

И затем используйте их в своих тестах:

<?php

$I = new TestGuy($scenario);

$I->seePageReloaded();

$I->seeClassIsLoaded('TestGuy');

$I->dontSeeUserExist($user);

?>

Каждая see или dontSee функция требует по крайней мере одно утверждение. Codeception использует утверждения из PHPUnit.

Вы можете объявлять утверждения используя asertXXX методы модуля.  
Codeception использует утверждения из PHPUnit. Таким образом, в случае когда вам не хватает некоторых утверждений, вы можете использовать статичные методы PHPUnit из класса PHPUnit\_Framework\_Assert.

<?php

function seeClassExist($class)

{

$this->assertTrue(class\_exists($class));

// или

\PHPUnit\_Framework\_Assert::assertTrue(class\_exists($class));

}

?>

В ваших помощниках вы можете использовать эти утверждения:

<?php

function seeCanCheckEverything($thing)

{

$this->assertTrue(isset($thing), "this thing is set");

$this->assertFalse(empty($any), "this thing is not empty");

$this->assertNotNull($thing, "this thing is not null");

$this->assertContains("world", $thing, "this thing contains 'world'");

$this->assertNotContains("bye", $thing, "this thing doesn`t contain 'bye'");

$this->assertEquals("hello world", $thing, "this thing is 'Hello world'!");

}

?>

Просто введите $this->assert чтобы увидеть их все.

Так же, у каждого модуля есть специальные методы $this->assert и $this->assertNot. Оба метода принимают одинаковые параметры и полезны если вам нужно объявить как положительное так и отрицательное утверждение в вашем модуле. Эти функции принимают массив как параметр, где первое значение массива это название функции утверждения из PHPUnit.

<?php

$this->assert(array('Equals',$int,3));

$this->assertNot(array('internalType',$int,'bool'));

$this->assert(array('Contains', array(3,5,9), 3));

?>

Давайте посмотрим как определить оба see и dontSee действия без дублирования кода.

assert($this->proceedSeeClassExist($class));

}

public function dontSeeClassExist($class)

{

$this->assertNot($this->proceedSeeClassExist($class));

}

protected function proceedSeeClassExist($class)

{

return array('True',get\_class($class));

}

?>

Для dontSeeClassExist, будет вызван assertFalse.

### Разрешение конфликтов

Что случится если у вас есть два модуля, которые содержат действия с одинаковыми названиями?  
Codeception позволяет вам переопределять действия меняя порядок модулей.  
Действие из второго модуля будет загружено, а действие из первого модуля будет проигнорировано.  
Порядок модулей описывается в файле настроек тестового набора.

### Взаимодействие Модулей

Возможно, что вы захотите получить доступ к внутренним свойствам или функциям из другого модуля. Например, для вашего модуля понадобилось соединение из модуля Doctrine или браузер из модуля Symfony.

Модули могут взаимодействовать с другими модулями через метод getModule. Пожалуйста, обратите внимание, что этот метод может выбросить исключение если требуемый модуль не был загружен.

Давайте представим, что мы пишем модуль, который переподключается к базе данных. Предположим он использует значение свойства dbh из модуля Db.

<?php

function reconnectToDatabase() {

$dbh = $this->getModule('Db')->dbh;

$dbh->close();

$dbh->open();

}

?>

Используя функцию getModule вы получаете доступ ко всем публичным методам и свойствам запрошенного модуля.  
Свойство dbh было специально объявлено как публичное чтобы быть доступным для других модулей.  
Эта техника так может быть полезна если вам нужно выполнить последовательность действий взятых из других модулей.  
Например:

<?php

function seeConfigFilesCreated()

{

$filesystem = $this->getModule('Filesystem');

$filesystem->seeFileFound('codeception.yml');

$filesystem->openFile('codeception.yml');

$filesystem->seeInFile('paths');

}

?>

### Неопределенные действия в помощниках

В случаи когда у вас есть действие, которое еще не определено, вы можете автоматически создать метод-заглушку для него в соответствующем помощнике. Чтобы это сделать, вы можете использовать команду analyse, которая сканирует все тесты и ищет несуществующие действия в любом подключенном модуле.

Таким образом, вы можете доверить написание тестов технически не подкованным людям или специалистам по качеству. В случае если им будет недоставать некоторых действий они объявят их в тесте.

<?php

$I->doManyCoolThings();

?>

Запустив команду analyze у вас спросят, хотите ли вы добавить doManyCoolThings в текущий Помощник.

### Хуки

Каждый модуль может обрабатывать события из запущенного теста. Модуль может быть выполнен до начала теста или после его завершения. Это может быть полезно для действий начальной загрузки или завершающих действий.  
Вы так же можете определить специальное поведение для ситуации когда тест провален. Это может быть полезно отладке.  
Например, модуль PhpBrowser сохраняет текущую страницу в директорию log если тест провален.

Все хуки определены в \Codeception\Module и перечислены здесь. Вы вольны переопределить их в своем модуле.

Пожалуйста, обратите внимание методы с префиксом \_ не добавлены в класс "Парня". Это позволяет объявлять публичные методы, но использовать только для внутренних целей.

### Отладка

Как мы уже упоминали, хук \_failed может помочь в отладке проваленного теста. У вас есть возможность сохранить текущее состояние тестов и показать их пользователю.

Вы не ограничены этим. Каждый модуль может выводить внутренние значения, которые могут быть полезны при отладке.  
Например, модуль PhpBrowser выводит http код ответа и текущий адрес каждый раз когда он переходит на новую страницу.  
Таким образом, модули не являются "черными ящиками". Они пытаются показать вам, что происходит в время теста. Это делает отладку ваших тестов менее болезненной.

Чтобы вывести дополнительную информацию, используйте debug и debugSection методы модуля.  
Вот пример того, как это работает для PhpBrowser:

<?php

$this->debug('Request ('.$method.'): '.$uri.' '. json\_encode($params));

$browser->request($method, $uri, $params);

$this->debug('Response code: '.$this->session->getStatusCode());

?>

Этот тест запущенный с помощью модуля PhpBrowser в режиме отладке, выведет что-то похожее на это:

I click "All pages"

\* Request (GET) http://localhost/pages {}

\* Response code: 200

### Настройка

Модули могут быть настроены в файле настроек тестового набора или глобально в codeception.yml.  
Обязательные параметры должны быть определены в $requiredFields свойстве класса модуля. Вот как это сделано в модуле Db.

<?php

class Db extends \Codeception\Module {

protected $requiredFields = array('dsn', 'user', 'password');

?>

Следующий раз вы когда запустите набор тестов без установки эти значений будет выброшено исключение.

Для необязательных параметров, вы должны установить значения по умолчанию. Свойство $config используется чтобы определить необязательные параметры, а так же их значения. В модуле Selenium мы используем адрес и порт сервера по умолчанию.

<?php

class Selenium extends \Codeception\Util\MinkJS

{

protected $requiredFields = array('browser', 'url');

protected $config = array('host' => '127.0.0.1', 'port' => '4444');

}

?>

Параметры адрес и порт могут быть переопределены в настройках набора тестов. Значения устанавливаются в секции modules:config файла настроек.

modules:

enabled:

- Selenium

- Db

config:

Selenium:

url: 'http://mysite.com/'

browser: 'firefox'

Db:

cleanup: false

repopulate: false

Обязательные и необязательные параметры могут доступны через свойство $config. Используйте $this->config['parameter'] чтобы получить значение параметра.

### Динамическая настройка

начиная с версии 1.6.2

Если вы хотите переопределить настройки модуля во время исполнения, вы можете использовать \_reconfigure метод модуля.  
Вы можете вызвать его из класса помощника и передать в него все поля, которые хотите изменить.

<?php

$this->getModule('Selenium2')->\_reconfigure(array('browser' => 'chrome'));

?>

В конце теста все ваши изменения откатятся до значений из файла настроек.

## МОДУЛЬ DB

При создании тестов которые взаимодействуют с базой данных, необходимо, чтобы база данных была подготовлена и удалены изменения внесенные предыдущими тестами. Это можно делать вручную или автоматически, что реализуется с помощью **модуля Db**, который так же создает подключение с БД и предоставляет свои методы тестирования данных.   
  
По умолчанию модуль будет пытаться заполнить базу данными из дампа, и очистить ее после выполнения каждого теста. А если точнее, он перезапишет данные в БД из подготовленного для этого дампа. Используется дамп данных в **SQL** формате.  
В настоящее время документация фреймворка для тестирования Codeception немного отстала от разработки самого проекта, поэтому в статье возможны неточности.  
  
Модуль Db работает с любыми базами данных поддерживающими **PDO**. Он может быть использован с любыми тестами, если они не являются довольно медленными. Загрузка дампа может занять продолжительное время, поэтому для данной цели могут быть использованы и другие подходы.  
  
Для использования модуля Db нужно:

* создать тестовую БД (в примере конфигурации это test.loc);
* заполнить ее данными необходимыми для тестов (если нужно) вручную или используя фикстуры;
* создать дамп этой базы и поместите его в папку tests/\_data (в примере это dump.sql);
* добавить модуль с нужными параметрами в конфигурационный файл.

Т.к. соединение модуля с базой данных может понадобиться для разных типов тестов, можно указать настройки соединения в основном конфигурационном файле **codeception.yml**:

modules:

config:

Db:

dsn: 'mysql:host=localhost;dbname=test'

user: 'root'

password: ''

а подключить сам модуль Db с указанием более специфичных настроек в конфигурационном файле соответствующего типа тестов, например в **tests/acceptance.suite.yml** (будем создавать приемочные тесты):

modules:

enabled:

- PhpBrowser:

url: http://test.loc

- \Helper\Acceptance

- Db:

dump: 'tests/\_data/test.sql'

populate: true

cleanup: true

reconnect: true

тут:  
- populate - следует ли загружать дамп перед запуском набора тестов  
- cleanup – восстанавливать ли БД из дампа перед каждым тестом  
- reconnect - должен ли модуль повторно подключаться к базе данных перед каждым тестированием  
  
Еще примеры настройки и использования данного модуля можно посмотреть в файле vendor\codeception\codeception\docs\modules\Db.md

##### Методы модуля Db.

Есть ряд методов которые можно использовать для тестирования при использовании модуля Db:  
`seeInDatabase`,` dontSeeInDatabase`, `seeNumRecords`,` grabFromDatabase` и `grabNumRecords`   
они принимают массивы в качестве критериев выборки. Условие WHERE генерируется с использованием ключа элемента в качестве имени поля и значение позиции в качестве значения поля.  
  
Пример:

$I->seeInDatabase ('users', array ('name' => 'Davert', 'email' => 'davert@mail.com'));

будет генерировать:

SELECT COUNT (\*) FROM `users` WHERE` name` = 'Davert' AND `email` = 'davert@mail.com'

Метод **grabColumnFromDatabase()** выбирает все значения из столбца в базе данных. Нужно указать имя таблицы, нужный столбец и критерии:

$mails = $I->grabColumnFromDatabase ('users', 'email', array ('name' => 'RebOOter'));

Метод **haveInDatabase()** добавляет запись в БД:

$I->haveInDatabase('users', array('name' => 'miles', 'email' => 'miles@davis.com'));

Метод **updateInDatabase()** обновляет запись в базе данных:

$I->updateInDatabase ('users', array ('isAdmin' => true), array ('email' => 'miles@davis.com'));

Примеры использования методов см. в файле vendor\codeception\codeception\docs\modules\Db.md  
  
  
Модуль Db содержит несколько публичных свойств, которые могут понадобиться для создания соединения с базой данных в своих тестах:

* dbh (содержит соединение PDO);
* driver (содержит драйвер подключения)

Получить доступ к этим свойствам можно из помощников, а уже к свойствам и методам помощников есть доступ непосредственно в тесте.

#### Фикстуры.

Для хранения фикстур используется класс **Fixtures** который представляет из себя хранилище данных. Сначала мы сохраняем туда нужные данные (массивы фикстур), а далее можем получить в любом нужном классе. Для удобства создан статический интерфейс методов класса Fixtures, т.е. не нужно получать объект.  
  
Методы класса Fixtures можно посмотреть в файле vendor\codeception\codeception\src\Codeception\Util\Fixtures.php  
Основные:

Fixtures::add('users', ['name' => 'Ivan']); //сохраняем

Fixtures::exists('users'); //проверяем наличие в хранилище

Fixtures::get('users'); //получаем

Согласно текущей документации, фикстуры должны быть заданы в **bootstrap** файле, т.е. единожды и до начала тестирования.   
  
Перед использованием фикстур нужно создать тестовую базу данных. Это делается вручную или используя модуль Db (перезагрузкой дампа).   
Обычно создается чистая тестовая база данных со структурой нужных таблиц, заполнение которых уже происходит с помощью фикстур.  
  
Вы можете использовать в классах **Cest** метод **\_before()** для загрузки фикстур перед началом теста (каждого метода). Метод **\_after** позволяет очистить БД между выполнением тестирующих методов одного класса. Удалять внесенные тестами данные после выполнения каждого теста можно и с помощью модуля Db, если в настройках модуля указать дамп и параметр cleanup: true

dump: 'tests/\_data/test.sql'

cleanup: true

Если быть точным, то данный механиз просто перезапишет базу данных из дампа. И так перед каждым тесто, что занимает сравнительно много времени. Поэтому в моем примере я не буду это использовать.  
  
Но подключить модуль Db для соединения с базой данных нужно в любом случае. Например так в файле **tests\acceptance.suite.yml**:

modules:

enabled:

- Db:

dsn: 'mysql:host=localhost;dbname=test'

user: 'root'

password: ''

В качестве примера создам приемочный тест, работающий с все той же таблицей users.   
  
Для хранения фикстур создаем папку **\_fixtures** в каталоге test. В ней создаем файл фикстур users.php который будет содержать данные для добавления в таблицу users:

<?php

return [

[

'name' => 'testName1',

'email' => 'testName1@gmail.com',

],

[

'name' => 'testName2',

'email' => 'testName2@gmail.com',

]

];

Для удобства воспользуемся файлом предзагрузки **tests\\_bootstrap.php**, создаем если его нет. Определяем константу которая будет содержать путь к папке с фикстурами:

<?php

define("FIXTURES\_DIR", \_\_DIR\_\_ . '/\_fixtures/');

Данная константа может понадобиться для различных типов тестов.  
  
Теперь создаем, файл предзагрузки для приемочных тестов **\_bootstrap.php** в папке **tests\acceptance** с содержимым:

<?php

use Codeception\Util\Fixtures;

/\*

\* Используем класс Fixtures для хранения фикстур

\* FIXTURES\_DIR - константа заданная в главном файле \_bootstrap

\*/

Fixtures::add('users', require(FIXTURES\_DIR . 'users.php'));

Данный код можно указать и в основном файле предзагрузки, созданном выше, если вы планируете использовать указанные фикстуры и для других типов тестов (функциональных и модульных).  
Теперь до начала тестирования у нас будет подключен файл фикстур и сохранен в глобальное хранилище фикстур которое предоставляет класс Fixtures.   
  
Создаем приемочный тест для демонстрации работы с фикстурами:

codecept generate:cest acceptance UserInDb

Добавляем содержимое:

<?php

use Codeception\Util\Fixtures;

class UserInDbCest

{

public function \_before(AcceptanceTester $I)

{

if(Fixtures::exists('users')){

/\*

\* Добавляем строки в таблицу 'users'

\*/

foreach (Fixtures::get('users') as $item) {

$I->haveInDatabase('users', $item);

}

}

}

public function \_after(AcceptanceTester $I)

{

}

public function checkUserInDb(AcceptanceTester $I)

{

$I->wantTo('Проверяем наличие данных в таблице users');

$I->seeInDatabase ('users', array ('name' => 'testName1', 'email' => 'testName1@gmail.com'));

}

}

После запуска этого теста:

codecept run acceptance UserInDbCest

должны получить уведомление об успешном его прохождении. Значит фикстуры отработали.  
  
В данном примере я использовал метод **\_before()** для заполнения таблицы фикстурами (нужными для тестирования данными). Напомню, что данный метод выполняется автоматически перед каждым тестовым методом.   
Строкой

if(Fixtures::exists('users'))

проверяем наличие нужной фикстуры в хранилище по ее ключу. Получаем данные с помощью

Fixtures::get('users')

и далее добавляем массив данных в БД используя метод haveInDatabase():

$I->haveInDatabase('users', $item);

Т.к. массив содержит вложенные массивы каждый из которых представляет собой отдельный набор для добавления в таблицу (отдельная строка) используется цикл.  
  
Видно, что я не использовал метод **\_after()** для очистки таблицы между тестами. Это не потребовалось т.к. я использовал метод **haveInDatabase()** для заполнения данными, а данный метод автоматически удаляет внесенные данные сразу после теста.

Перед каждым запуском нужно заполнять базу тестовыми данными. Для этих вещей можно подключить и настроить модуль Db для импорта дампа, но фреймворк предоставляет работу с тестовыми данными в виде простых PHP-массивов.

Для модульных и интеграционных тестов расширение yii2-codeception поддерживает фикстуры «из коробки». Другое дело – - функциональные и приёмочные. Для поддержки в них миграций можно пойти путём написания собственного плагина для Codeception. Мы, возьмём готовый [FixtureHelper](https://github.com/yiisoft/yii2-app-advanced/blob/master/tests/codeception/common/_support/FixtureHelper.php) из advanced-приложения, поместим его в папку tests/codeception/\_support и поменяем лишь пути и пространство имён:

namespace tests\codeception\\_support;

*// use tests\codeception\fixtures\UserFixture;*use Codeception\Module;

use yii\test\FixtureTrait;

use yii\test\InitDbFixture;

**class** FixtureHelper **extends** Module{

use FixtureTrait {

loadFixtures **as** **protected**;

fixtures **as** **protected**;

globalFixtures **as** **protected**;

unloadFixtures **as** **protected**;

getFixtures **as** **protected**;

getFixture **as** **protected**;

}

**public** **function** \_beforeSuite($settings = [])

{

$this->loadFixtures();

}

**public** **function** \_afterSuite()

{

$this->unloadFixtures();

}

**public** **function** globalFixtures()

{

**return** [

InitDbFixture::className(),

];

}

**public** **function** fixtures()

{

**return** [

*//'user' => [*

*// 'class' => UserFixture::className(),*

*// 'dataFile' => '@tests/codeception/fixtures/data/user.php',*

*//],*

];

}}

Фикстуру UserFixture мы пока закомментировали. Скоро мы её добавим свою.