# 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

## Инициализация окружения для тестирования Codecept

Сodecept bootstrap

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

Выполните следующую команду для генерации структуры каталогов для тестов и вспомогательных файлов:

codecept bootstrap -a Engineer app/tests/codeception

Примечание

Тут же рекомендую ознакомится с руководством http://allframeworks.ru/post/codeception-2-0-alpha.html, которое дает понимание по выбору актера на запрос: "Select an actor. Default: Guy". Вместо Engineer вы вправе указать подходящее для вашего случая имя актера, например: Client, Manager, Admin, ApiClient.

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 для тестов файлы..

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

├── \_helpers

│ ├── CodeHelper.php

│ ├── TestHelper.php

│ └── WebHelper.php

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

├── report.json

├── report.tap.log

└── ManageUserCest.create.fail.png

#### Настройка.

При инициализации Codeception в корне приложения появится главный конфигурационный файл **codeception.yml**. В нем указаны общие настройки для тестирования и можно дополнительно указать глобальные модули и их параметры, которые нужны для разных типов тестов, например:

modules:

config:

Db:

dsn: ''

user: ''

password: ''

Для каждого типа тестов (приемочные, функциональные, модульные) существует дополнительный, специализированный конфигурационный файл. Например для проведения приемочного тестирования нужно настроить общий конфигурационный файл - **tests\acceptance.suite.yml** указать домен сайта для **PhpBrowser**:

actor: AcceptanceTester

modules:

enabled:

- PhpBrowser:

url: http://test.loc

- \Helper\Acceptance

- Db:

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

populate: true

cleanup: true

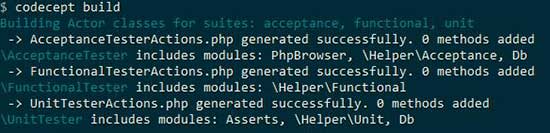
reconnect: true

Так же, в примере данного конфигурационного файла видим подключение модуля Db с дополнительными настройками, основные настройки которого указаны выше в главном конфигурационном файле.  
  
Можно вывести в консоль массив текущей конфигурации (какие указаны настройки, какие можно указать)

codecept config:validate

После внесения правок в конфигурационные файлы нужно выполнить команду:

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

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

**<?php**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');

Конфигурация

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.

По умолчанию главный файл конфигурации выгляди приблизительно вот так:

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: ''

user: ''

password: ''

dump: tests/\_data/dump.sql

Конфигурация для приемочных (acceptance) тестов acceptance.yml

class\_name: WebGuy

modules:

# enabled modules and helpers

enabled:

- PhpBrowser

- WebHelper

- Db

# local module configuration. Overrides the global.

config:

Db:

dsn:

## 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.

php codecept.phar 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 комманду

php codecept.phar 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.

php codecept.phar 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

Изначально нас интересуют эти файлы:

tests

└── codeception

├── acceptance

│ └── \_bootstrap.php

├── functional

│ └── \_bootstrap.php

└── unit

└── \_bootstrap.php

Файлы acceptance/\_bootstrap.php и functional/\_bootstrap.php имеют внутри себя код запуска приложения с соответствующим конфигурационным файлом:

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

и:

**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.

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

Далее рассмотрим именно настройки Codeception:

tests

├── codeception

│ ├── acceptance.suite.yml

│ ├── functional.suite.yml

│ └── unit.suite.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

Файл unit.suite.yml весьма скромен со своей единственной строкой:

class\_name: UnitTester

Оставляем его тоже.

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, как и говорили на вебинаре, 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.

Теперь подключим этот модуль 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'

Аналогично добавим его 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.

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 пооддерживающими **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 помошников, а уже к свойствам и методам помошников есть доступ непосредственно в тесте.  
  
Для ускорение выполнения тестов взаимодействующих с БД рекомендуется использование базы данных **SQLite**.

#### Фикстуры.

По фикстурам в Codeception (без использования интеграции с фреймворками) пока обновленной информации совсем мало. Та, что есть в официальной документации, устарела. Поэтому опишу свое видение и в будущем, возможно, откорректирую.  
  
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.

##### 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()**.