We do IT to make you smile!

Test variants

Functional tests

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- Smoke Testing
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing
- Negative Testing

Non-functional tests

- Performance Testing
- Load Testing
- Stress Testing
- Volume Testing
- Security Testing
- Compatibility Testing
- Install Testing
- Recovery Testing
- Reliability Testing
- Usability Testing
- Compliance Testing
- Localization Testing

Test variants

- Functional tests
  - **Unit Testing**
  - Integration Testing
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - Beta/Acceptance Testing
  - Negative Testing

Testing of an individual software component or module is termed as Unit Testing. It is typically done by the PROGRAMMER and not by testers, as it requires detailed knowledge of the internal program design and code. It may also require developing test driver modules or test harnesses.

Test variants

- **Functional tests**
  - Unit Testing
  - **Integration Testing**
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - Beta/Acceptance Testing
  - Negative Testing

Testing of all integrated modules to verify the combined functionality after integration is termed as Integration Testing.

Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

Test variants

- **Functional tests**
  - Unit Testing
  - Integration Testing
  - **System Testing**
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - Beta/Acceptance Testing
  - Negative Testing

Under System Testing technique, the entire system is tested as per the requirements. It is a Black-box type Testing that is based on overall requirement specifications and covers all the combined parts of a system.

Test variants

Functional tests

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- **Smoke Testing**
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing
- Negative Testing

Whenever a new build is provided by the development team then the Software Testing team validates the build and ensures that no major issue exists.

The testing team ensures that the build is stable and a detailed level of testing is carried out further. Smoke Testing checks that no show stopper defect exists in the build which will prevent the testing team to test the application in detail.

If testers find that the major critical functionality is broken down at the initial stage itself then testing team can reject the build and inform accordingly to the development team.

Test variants

- Functional tests
  - Unit Testing
  - Integration Testing
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - **Regression Testing**
  - Beta/Acceptance Testing
  - Negative Testing

Testing an application as a whole for the modification in any module or functionality is termed as Regression Testing. It is difficult to cover all the system in Regression Testing, so typically Automation Testing Tools are used for these types of testing.

Test variants

- **Functional tests**
  - Unit Testing
  - Integration Testing
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - **Beta/Acceptance Testing**
  - Negative Testing

Beta Testing is a formal type of Software Testing which is carried out by the customer. It is performed in the Real Environment before releasing the product to the market for the actual end-users.

Beta Testing is carried out to ensure that there are no major failures in the software or product and it satisfies the business requirements from an end-user perspective. Beta Testing is successful when the customer accepts the software.

Test variants

- **Functional tests**
  - Unit Testing
  - Integration Testing
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - **Beta/Acceptance Testing**
  - Negative Testing

An Acceptance Test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end-user. Client accepts the software only when all the features and functionalities work as expected.

It is the last phase of the testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

Test variants

- **Functional tests**
  - Unit Testing
  - Integration Testing
  - System Testing
  - Sanity Testing
  - Smoke Testing
  - Interface Testing
  - Regression Testing
  - Beta/Acceptance Testing
  - **Negative Testing**

Testers having the mindset of "attitude to break" and using Negative Testing they validate that if system or application breaks. A Negative Testing technique is performed using incorrect data, invalid data or input. It validates that if the system throws an error of invalid input and behaves as expected.

Test variants

This term is often used interchangeably with ‘STRESS' and ‘LOAD' testing. Performance Testing is done to check whether the system meets the performance requirements. Different performance and load tools are used to do this testing.

- Non-functional tests
  - **Performance Testing**
  - Load Testing
  - Stress Testing
  - Volume Testing
  - Security Testing
  - Compatibility Testing
  - Install Testing
  - Recovery Testing
  - Reliability Testing
  - Usability Testing
  - Compliance Testing
  - Localization Testing

Test variants

Load Testing helps to find the maximum capacity of the system under specific load and any issues that cause software performance degradation. Load testing is performed using tools like JMeter, LoadRunner, WebLoad, Silk performer, etc.

- Non-functional tests
  - Performance Testing
  - **Load Testing**
  - Stress Testing
  - Volume Testing
  - Security Testing
  - Compatibility Testing
  - Install Testing
  - Recovery Testing
  - Reliability Testing
  - Usability Testing
  - Compliance Testing
  - Localization Testing

Test variants

This testing is done when a system is stressed beyond its specifications in order to check how and when it fails. This is performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to the system or database load.

- Non-functional tests
  - Performance Testing
  - Load Testing
  - **Stress Testing**
  - Volume Testing
  - Security Testing
  - Compatibility Testing
  - Install Testing
  - Recovery Testing
  - Reliability Testing
  - Usability Testing
  - Compliance Testing
  - Localization Testing

Test variants

Security Testing is done to check how the software or application or website is secure from internal and external threats. This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are.

It also checks how software behaves for any hackers attack and malicious programs and how software is maintained for data security after such a hacker attac

- Non-functional tests
  - Performance Testing
  - Load Testing
  - Stress Testing
  - Volume Testing
  - Security Testing
  - Compatibility Testing
  - Install Testing
  - Recovery Testing
  - Reliability Testing
  - Usability Testing
  - Compliance Testing
  - Localization Testing

Manual vs Automated

Manual testing is done in person, by clicking through the application or interacting with the software and APIs with the appropriate tooling. This is very expensive as it requires someone to set up an environment and execute the tests themselves, and it can be prone to human error as the tester might make typos or omit steps in the test script.

Automated tests, on the other hand, are performed by a machine that executes a test script that has been written in advance. These tests can vary a lot in complexity, from checking a single method in a class to making sure that performing a sequence of complex actions in the UI leads to the same results.

Automated testing is a key component of continuous integration and continuous delivery and it's a great way to scale your QA process as you add new features to your application.

Codeception: PHP testing framework

- Allows you to write compact, simple and easy to read tests
- Allows creation of acceptance, functional and unit tests
- Tests can be grouped in scenarios that run together as a block
- Supports all major frameworks: Sympfony, Laravel, Yii Framework, Phlacon, Wordpress, Joomla
- Gives you the ability to reuse parts of your code for multiple tests

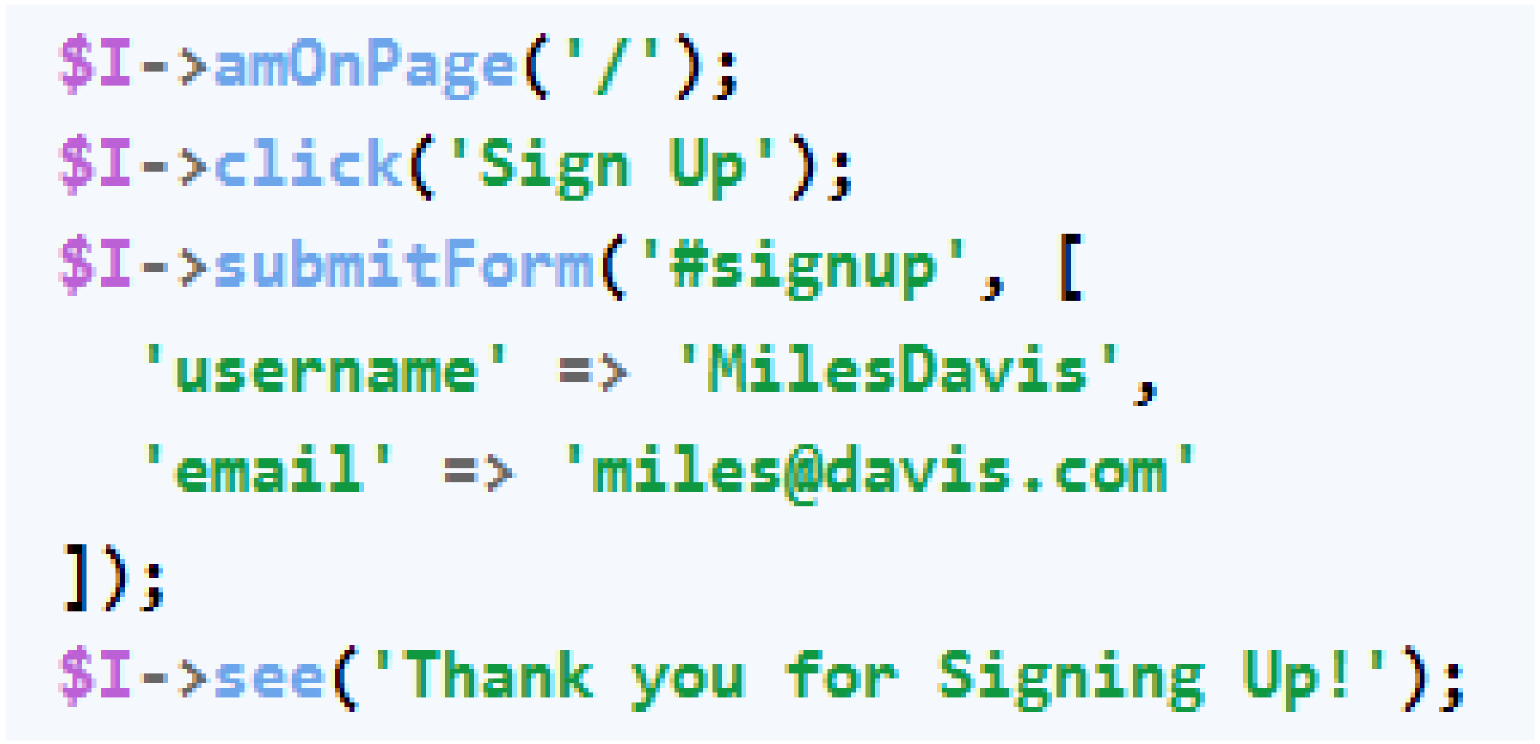
Install Codecption

- Install via composer
- Install by cloning the git repo
- Install by downloading codecept.phar
  - php codecept.phar bootstrap **// initializes codeception folder infrastructure**

Codeception: test types

	Unit tests	Functional tests	Acceptance tests
Scope	Single PHP class	PHP framework (Routing, Controller, API, etc.)	Page in browser (Chrome, Firefox, PhpBrowser headless)
Test needs access to PHP code	**Yes**	**Yes**	**No**
Webserver required	**No**	**No**	**Yes**
JavaScript testable	**No**	**No**	**Yes**
Additional software required	**None**	**None**	**Selenium server for browser emulation**
Speed	**Fast**	**Fast**	**Slow**
Configuration file	**unit.suite.yml**	**functional.suite.yml**	**acceptance.suite.yml**

Codeception: Acceptance test

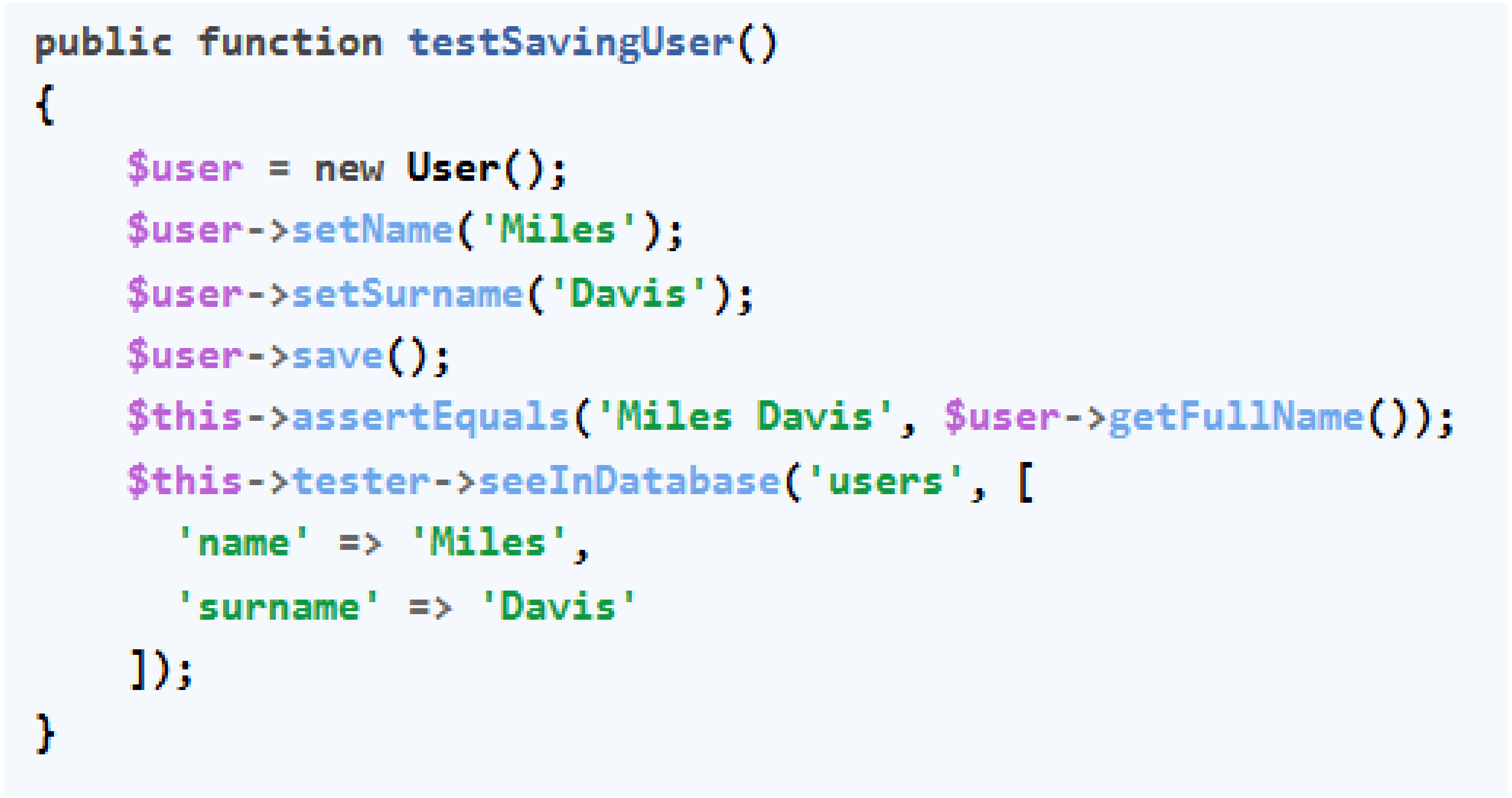


```
$I->amOnPage('/');
$I->click('Sign Up');
$I->submitForm('#signup', [
   'username' => 'MilesDavis',
   'email' => 'miles@davis.com'
]);
$I->see('Thank you for Signing Up!');
```

Codeception: Functional tests

```php
$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']);
```

Codeception: Unit tests



```php
public function testSavingUser()
{
    $user = new User();
    $user->setName('Miles');
    $user->setSurname('Davis');
    $user->save();
    $this->assertEquals('Miles Davis', $user->getFullName());
    $this->tester->seeInDatabase('users', [
      'name' => 'Miles',
      'surname' => 'Davis'
    ]);
}
```

Codeception: Reusability

- **Actors**: Use actor classes to set common actions which can be used across a suite. This is the proper way to extend the base performable actions of Codeception.

- **PageObjects**: represents a web page as a class and the DOM elements on that page as its properties, and some basic interactions as its methods. It can be shared between acceptance and functional tests.

```
php vendor/bin/codecept generate:pageobject acceptance Login
```

- **StepObjects**: it serves as extension for **ActorObjects**. Specific for a role functiions can be implemented in a **StepObject**

Codeception: Modules, Helpers, Hooks

- **Modules**: All actions and assertions that can be performed by the Tester object in a class are defined in modules.

- **Helpers**: give the ability to write custom actions, that can be used by the suite for which the **Helper** is implemented.

- **Hooks:** give the ability to write custom actions, that can be used by the suite for which the Helper is implemented.

Codeception: customizations

- **Create custom boostrap_autload.php file:**

  ```
  <?PHP
  \CODECEPTION\UTIL\AUTOLOAD::ADDNAMESPACE('NAME\SPACE', 'SRC\PATH');
  ```

- **Add the newly created file to the codeception global configuration:**

  ```
  ...
  bootstrap: boostrap_autload.php
  ....
  ```

References

- https://www.guru99.com/types-of-software-testing.html
- https://www.softwaretestinghelp.com/types-of-software-testing
- https://codeception.com/docs/reference/Commands