# Tests unitaires - PHP Unit – Suite de tests – Configuration – Doubles – Couverture de code

**Objectifs** :

L’objectif de cette séance est de compléter les fonctionnalités et usage des tests unitaires , à partir de la librairie la plus utilisée de PHP pour les tests unitaires, PHPUnit.

Nous allons aborder l’installation via Composer et des tests simples de code retour d’une Api Web.

Nous verrons différentes techniques de double permettant d’isoler nos tests mais également de simuler des comportements, tout en restant à un niveau de tests …. Unitaires.

Nous verrons plus tard dans l’année d’autres librairies de test, dans le domaine Java.

## Test d’une API- Installations via Composer

Nous avons installé PhpUnit de manière globale la dernière fois. Cette fois ci nous allons l’installer via l’outil de gestion de package de PHP, beaucoup d’entre vous confondent et méconnaissent ces outils.

Nous allons également dans le même projet installer Guzzle, client http très utilisé pour accéder à des API Web.

Puis nous allons utiliser un fichier de configuration XML pour définir nos tests XML.

Enfin nous allons effectuer quelques tests de connexion à l’API via Guzzle.

1. Créez un nouveau répertoire (dans Laragon\www)
2. Sous Packagist, cherchez les packages suivant : PHPUnit et Guzzle.

<https://packagist.org/>

1. Exécutez les commandes indiquées pour acquérir ces packages dans le répertoire que vous avez défini (en a. )
2. Exécutez ensuite la commande : composer update
3. Copiez le fichier **phpunit.xml** fourni à la racine de ce répertoire
4. Ouvrez ce fichier. Dans quel répertoire va-t-il chercher des tests unitaires à lancer ? Créez le répertoire manquant.
5. Dans le nouveau répertoire, copiez le fichier php fourni nommé testRequestService.php
6. Avec votre navigateur, allez sur le site <https://httpbin.org/> puis sur l’URL user-agent et avec l’outil de débogage CHROME, examinez les entêtes et le code retour

* Dans la partie Network 🡪 Headers

1. A la racine du répertoire de tests, exécutez phpunit et expliquez les résultats

Question 1 : faites une recherche et documentez l’intérêt des méthodes setUp() et tearDown()

Exercice 1 : complétez le test testPut avec une assertion comparant le code statut avec la valeur attendue.

Exercice 2 : écrire un nouveau test pour la récupération d’image

On veut tester 2 choses : la bonne récupération d’une image, au format jpeg.

Principes : tout d’abord, cherchez l’URL sur le site httpbin.org permettant de récupérer aléatoirement une image.

Dans le test, vous préciserez dans la requête Guzzle le format attendu de l’image, par exemple jpeg.

<http://docs.guzzlephp.org/en/stable/request-options.html>

Remarque : on peut spécifier ce genre d’options Guzzle soit au niveau global, dans le setup, soit requête par requête : c’est ce qu’il faut faire ici pour ne pas casser les autres tests.

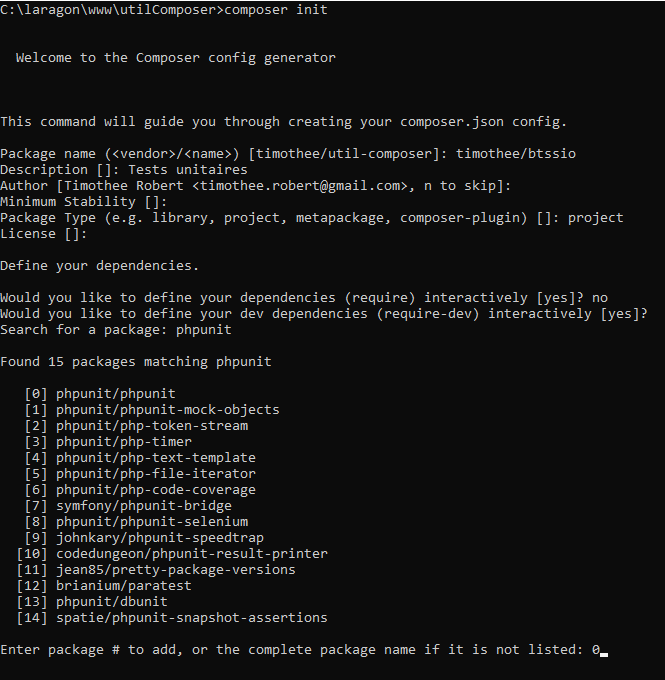
## Projet Composer – tests divers avec PHPUnit

Nous allons aborder une configuration de projet avec Composer et effectuer différents tests utilisant PHPUnit.

1. Créer un nouveau répertoire et aller dans ce répertoire en ligne de commande (cd …)
2. Générer un projet via composer

Tapez la commande **composer init**

Cf ci-dessous. On définit une dépendance que pour le développement, les tests unitaires.



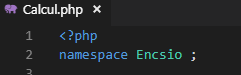
**Question** :

Quel fichier a été généré à l’issue de cette étape ?

La commande composer init a servi à générer ce fichier interactivement. Il aurait pu être créé indépendamment.

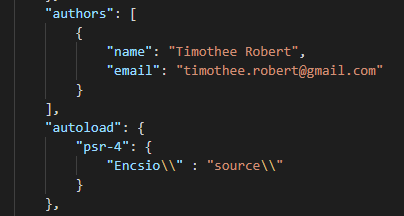
1. Tapez la commande **composer install**
2. Créez un répertoire **source** à la racine du projet, puis un fichier Calcul.php et une classe php Calcul
3. Dans ce fichier

* Déclarez un namespace.



* Ecrivez une fonction statique **nombreCarre** qui renvoie le carré d’un nombre passé en paramètre

1. Dans le fichier composer.json, ajout d’une entrée **autoload**



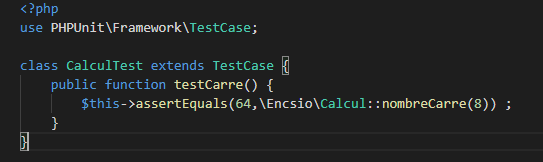
L’autoloading en PHP est très utile dans les projets d’une certaine taille pour éviter les *require*, il permet l’auto chargement des classes en allant chercher dans un ensemble de répertoires spécifiés, associés à un espace de nom.

Psr-4 est une convention de nommage des fichiers, classe etc en PHP : <https://www.php-fig.org/psr/psr-4/>

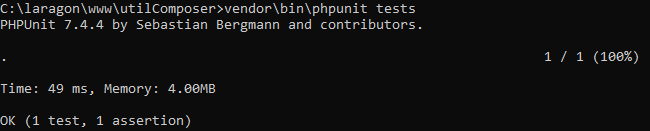
1. Mettre à jour l’autoloading



1. Créez un répertoire tests
2. Dedans, créer un fichier CalculTest.php avec le contenu suivant



1. Exécutez le test



Exercice 3 : trouvez un moyen de pouvoir dorénavant exécuter tous les tests dans le répertoire tests avec l’unique commande **phpunit**

## Couverture de code

La notion de couverture de code permet de savoir, pour un ensemble de classes données, quelle est le taux des fonctions développées couvertes par un (ou plusieurs) tests unitaires.

L’objectif est d’avoir 100 % de couverture de code pour un projet de développement. Il s’agit évidemment d’un minimum car la couverture de 100% signifie juste que chaque ligne de code des classes développées aura été accédée au moins une fois par un test.

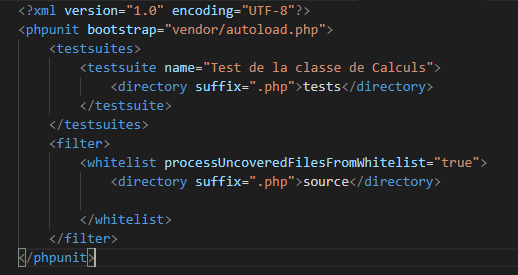
Nous conservons l’environnement de la partie 2 que nous allons compléter.

1. Installation de Xdebug

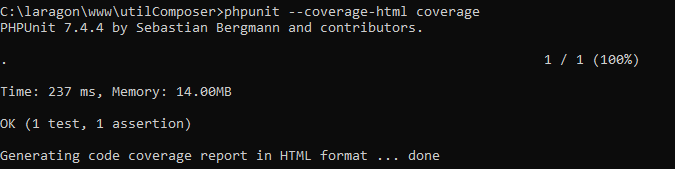
Pour ceux qui sont en PHP 7.1 x64 Thread Safe, comme moi, copiez la DLL fournie php\_xdebug-2.6.1-7.1-vc14-x86\_64.dll dans le répertoire : C:\laragon\bin\php\php-7.1.14-Win32-VC14-x64\ext

Pour les autres, téléchargez la bonne DLL sur le site Xdebug : <https://xdebug.org/download.php>

1. Dans Laragon, activez l’extension XDebug et redémarrez
2. Complétez le fichier phpunit.xml par un filtre et une liste de fichiers à parcourir, comme ceci :



1. En ligne de commande, tapez



**Checkpoint** : dans le répertoire coverage, vous devez trouver les fichiers HTML donnant l’analyse de couverture de code actuel de notre projet.

**Exercice** : test unitaire et couverture de code

1. ajoutez le fichier PHP Eleve.php dans le répertoire source du projet
2. écrire une classe de tests nommée EleveTest.php dans le répertoire tests

Dans le setUp, vous pouvez créez un objet de type Eleve, par exemple avec 3 matieres et 3 notes.

Astuce : fonction array pour passer directement un tableau en paramètre.

Ecrire une fonction qui va tester que le nombre de matières (ou notes) est bien 3.

Ecrire une fonction qui va tester que le calcul de la moyenne est correct.

En profiter pour corriger les 2 petites erreurs de la classe Eleve ….

## Pokemons: test unitaire – approche moderne et complète

Le problème avec notre approche naïve est que nous ne maîtrisons pas l’API Pokemon. D’ailleurs elle change souvent d’adresse, est parfois cassée.

Nous savons juste que la structure de l’API n’évolue pas, ou alors, quand elle évolue, elle assure la compatibilité.

Nous voulons cependant développer notre application autour de Pokemon et pouvoir la tester, dans tous ses aspects.

Comment faire alors que nous ne maîtrisons pas la disponibilité de l’API sous-jacente ?

La réponse : **utiliser des doubles**.

1. Modification du programme

La nécessité de pouvoir tester ses programmes dans la durée oblige un refactoring conséquent de notre classe d’origine.

* Comparez les programmes ***035\_Pokemon.php*** et ***040\_PokemonWrapper.php*** et listez les différences.

1. Utilisation de doubles

Afin de pouvoir s’affranchir de la disponibilité de l’API Pokemon, nous allons utiliser ce qu’on appelle dans le monde des tests des ***doubles***.

* En vous inspirant du test réalisé, ajoutez un nouveau test pour s’assurer de la bonne implémentation de la méthode de calcul du nombre de statistiques.

Astuce : reprenez le programme effectué pour récupérer des caractéristiques de Pokemon en Vue.js. Pour observer la réponse dans la console de débogage au format JSON, vous pouvez utiliser par exemple :

console.log(JSON.stringify(this.pokemons.stats, null, " ")) ;