UT03 Programación basada en lenguaje de marcas con código embebido - COMPOSER

Índice

Este material está basado en:

1. Libro de título: Desarrollo web en Entorno Servidor, de Juan Luis Vicente Carro, Editorial Garceta.

2. https://cizacas.github.io/DWES2425/

3. https://www.php.net/manual/es/index.php

Introducción

Composer es el gestor de dependencias para PHP. https://getcomposer.org/

Packagist es el repositorio de paquetes de Composer, ahí se publican los paquetes de PHP. https://packagist.org/

Gracias a Composer podemos instalar paquetes de terceros en nuestro proyecto.

Instalar Composer en Windows (globalmente)

Simplemente, descarga y ejecuta el instalador, https://getcomposer.org/download/ una vez lo tengas instalado en tu equipo podrás ejecutar el comando composer y todo estará funcionando. También es posible mediante comandos.

```
curl -sS https://getcomposer.org/installer | php
```

Ejecutar composer en nuestro proyecto

La siguiente sentencia nos muestra la versión de php instalada

```
php composer.phar -V
```

Añadir composer a un proyecto

Si hemos realizado una instalación a nivel global simplemente llamamos a **composer init,** si es una instalación local tendríamos que ejecutar:

```
php composer.phar
```

Instalacion Global

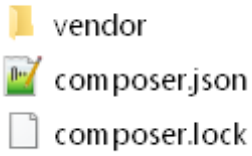
```
composer init

# Package name (<vendor>/<name>) [user/01-primer-proyecto]:
# Description []: Ejercicio 1 Composer
# Author[]:  Oscar <oscar@educantabria.es>
# Minimum Stability []: stable
# Package Type (e.g. library, project, metapackage, composer-plugin) []:
project
```

```
# License []:

Define your dependencies.

# Would you like to define your dependencies (require) interactively
[yes]? no
# Would you like to define your dev dependencies (require-dev)
interactively [yes]? no
# Add PSR-4 autoload mapping? Maps namespace "user/01-primer-proyecto" to
the entered relative path. [src/, n to skip]: no
```

Nos ha creado un archivo **composer.json** donde figuran las dependencias

```
{
    "name": "oscar/app-composer1",
    "description": "Ejercicio 1 Composer",
    "type": "project",
    "authors": [
        {
            "name": "Oscar",
            "email": "oscar@educantabria.es"
        }
    ],
    "require": {}
}
```

Creando la siguiente estructura dentro del proyecto:



Ejemplo de instalar una dependencia de una librería

Instalamos una librería que genera claves seguras Password Generator Library

https://github.com/hackzilla/password-generator

Antes comprueba que en c:\xampp\php\php.ini están habilitadas las siguientes extensiones:

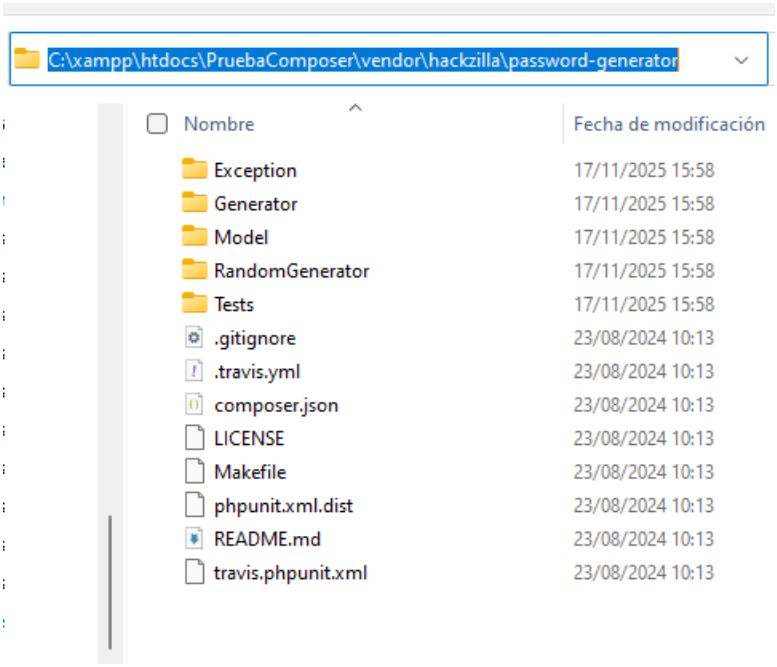
```
; habilitar extensión openssl
extension=openssl

; habilitar extensión zip
extension=zip
```

Y vamos a terminal para instalar la dependencia

```
composer require hackzilla/password-generator
```

En el composer.json incluye la dependencia

```
{
    "name": "oscar/prueba-composer",
    "authors": [
        {
            "name": "Oscar"
        }
    ],
    "require": {
        "hackzilla/password-generator": "^1.7"
    }
}
```



El archivo composer.lock

El archivo composer.lock es un archivo generado automáticamente por Composer que contiene información detallada sobre las dependencias específicas de un proyecto, incluyendo las versiones exactas de las dependencias instaladas. Este archivo se utiliza para garantizar que todos los desarrolladores en un equipo estén trabajando con la misma versión de las dependencias.

Cuando se ejecuta el comando “composer install” en un proyecto, Composer lee el archivo composer.json para determinar qué dependencias se deben instalar y en qué versiones. Luego, genera el archivo composer.lock, que contiene información detallada sobre las versiones exactas de las dependencias que se instalaron.

En general se recomienda no modificar el archivo composer.lock manualmente, ya que esto puede causar problemas de compatibilidad y conflictos de versiones. En lugar de eso, se recomienda utilizar el comando **“composer update”** para actualizar las dependencias de un proyecto, y dejar que Composer se encargue de actualizar el archivo composer.lock automáticamente.

Es una buena práctica subir a nuestros repositorios los archivos composer.json y composer.lock.

En resumen, el archivo composer.lock ayuda a garantizar la estabilidad de un proyecto al especificar las versiones exactas de las dependencias que se han utilizado.

Ejemplo de utilizar las dependencias instaladas

En el directorio crearemos un index.php

```php
<?php
require_once __DIR__ . '/../vendor/autoload.php';

use Hackzilla\PasswordGenerator\Generator\ComputerPasswordGenerator;

// crea el objeto
$generator = new ComputerPasswordGenerator();

// opciones para generar la contraseña
$generator
    ->setOptionValue(ComputerPasswordGenerator::OPTION_UPPER_CASE, true)
    ->setOptionValue(ComputerPasswordGenerator::OPTION_LOWER_CASE, true)
    ->setOptionValue(ComputerPasswordGenerator::OPTION_NUMBERS, true)
    ->setOptionValue(ComputerPasswordGenerator::OPTION_SYMBOLS, true)
    ->setLength(8);

$password = $generator->generatePassword();

echo $password;
?>
```

Instalar dependencias de desarrollo

Las dependencias de desarrollo son dependencias que sólo necesitamos en entornos de desarrollo, faker, phpunit, etc. Para añadir dependencias de desarrollo, usamos el parámetro `--dev` o `-d`:

```
composer require phpunit/phpunit --dev
```

Su uso es el mismo que siempre:

```php
require_once __DIR__ . '/vendor/autoload.php';

use PHPUnit\Framework\TestCase;

class Test extends TestCase
{
    public function test()
    {
        $this->assertTrue(true);
    }
```

```
}
```

En entornos de producción, cuando instalemos las dependencias, para evitar las de desarrollo, deberemos hacer lo siguiente:

```
composer install --no-dev
```

De esta forma sólo instalaremos dependencias de producción.

Actualizar dependencias

El comando es composer update

```
composer update phpunit/phpunit
```

Eliminar dependencias

El comando es **composer remove**

```
composer remove phpunit/phpunit
```

Ejemplo de crear nuestros scripts

Podemos crear nuestros scripts utilizando composer de la siguiente forma:

Es una etiqueta **scripts** en el composer.json

Por ejemplo, si queremos por ejemplo que se autoarranque un servidor para probar nuestra aplicación en el directorio `public`, añadimos al fichero **composer.json**

```
"scripts": {
        "start": "php -S localhost:8001 -t public"
    }
```

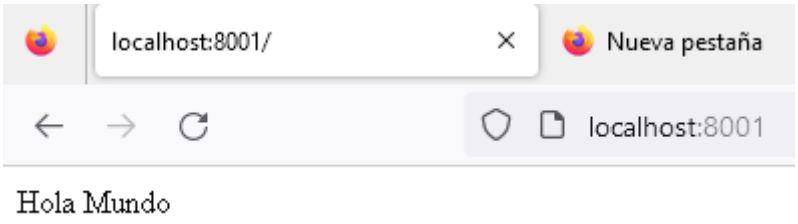
Crearemos el directorio `public` y pondremos la página web `index.php`, simplemente con un `echo`

```
echo "Hola Mundo";
```

A continuación, nos vamos a la terminal posicionados en nuestro directorio de trabajo y con el comando **composer start** pondremos activo el servidor

```
composer start
```

y desde cualquier navegador



Ejemplo de autocarga de los namespaces con PSR-4

PSR-4, es una especificación para la autocarga de clases desde la ruta donde se encuentran los archivos. Por ejemplo, queremos crear un directorio `app` y mapearlo con el espacio de nombres `App`. Para generar el namespace en el fichero **composer.json**

```
"autoload": {
      "psr-4": {
          "App\\": "app/"
      }
   }
```

En este caso hemos añadido la sección `autoload` y dentro de ella la sección `psr-4` que nos permite definir los namespaces que queremos utilizar en nuestro proyecto.

A continuación, debemos ejecutar el comando `composer dump-autoload` para que se generen los ficheros necesarios para que composer pueda cargar los namespaces.

```
   composer dump-autoload
```

Ahora vamos a crear los archivos y directorios necesarios para poder utilizar los namespaces. Por ejemplo, dentro del directorio `app` creamos un subdirectorio

Controllers donde vamos a crear el archivo `HolaMundoController.php` que contenga la una clase `HolaMundoController` con un método `index()`:

```
    namespace App\Controllers;

class HolaMundoController
{
    public function index(): void
    {
        echo "Hola Mundo segundo!";
    }
}
```

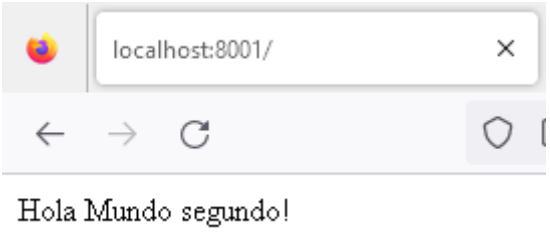
Ahora vamos a modificar el archivo `public/index.php` que será el punto de entrada de nuestra aplicación eliminando lo anterior:

```
// autoload composer
require_once __DIR__ . '/../vendor/autoload.php';

use App\Controllers\HolaMundoController; // el namespace

$holaMundoController = new HolaMundoController();

$holaMundoController->index();
```

Una vez tenemos todo preparado, podemos ejecutar el servidor web y comprobar que todo funciona correctamente.

```
composer dump-autoload
composer start
```

y desde cualquier navegador



Ejemplo autocarga de archivos independientes

Podemos generar namespaces para archivos independientes utilizando composer de la siguiente forma:

Por ejemplo, en el directorio `app` definido en el apartado anterior queremos cargar el archivo `helpers.php`. Modificamos el fichero **composer.json** incorporando:

```
"autoload": {
        "psr-4": {
            "App\\": "app/"
        },
        "files": [
            "app/helpers.php"
        ]
    }
```

En este caso hemos añadido la sección `autoload` y dentro de ella la sección `files` que nos permite definir los archivos que queremos utilizar en nuestro proyecto.

Ahora vamos a crear el archivo `helpers.php` en la carpeta `app` y vamos a añadir el siguiente código php:

```
    if ( ! function_exists('holaMundo')) {
    function holaMundo(): string
    {
        return 'Hola Mundo';
    }
}
```

A continuación, debemos ejecutar el comando `composer dump-autoload` para que se generen los ficheros necesarios para que composer pueda cargar los archivos.

```
composer dump-autoload
```

Ahora vamos a modificar en el archivo `public/index.php` incorporando:

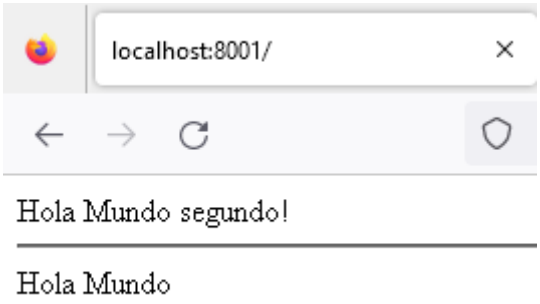
```
echo '<hr>';
```

```
echo holaMundo();
```

Una vez tenemos todo preparado, podemos ejecutar el servidor web y comprobar que todo funciona correctamente.

```
composer start
```

y desde cualquier navegador



Ejemplo Cargar una librería desde un directorio local

Si previamente hemos descargado la libreria de github a un directorio local utiizando el `comando clone` y queremos que en un proyecto se utilice esa libreria local debemos modificar el fichero **composer.json**

```
"repositories": [
        {
            "type": "path",
            "url": "Libreria"
        }
    ],
    "require":{
        "hackzilla/password-generator": "^1.6"
    }
```

En este caso hemos descargado la libreria externa en un directorio denominado `libreria` este directorio se encuentra dentro del proyecto definido y

en este caso para realizar la carga de la libreria hay que realizar un `composer update` si el proyecto ya existía o un `composer install` si es lo primero que realizamos.

```
composer update
```

Comandos avanzados

Composer ofrece una serie de comandos avanzados que pueden ser útiles en ciertas circunstancias:

- **composer show**: Este comando muestra información sobre las dependencias instaladas en un proyecto, incluyendo sus nombres, versiones y autores.
- **composer validate**: Este comando valida el archivo “composer.json” para asegurar que está bien formado y que las dependencias especificadas son válidas.
- **composer why-not<package>**: Este comando muestra por qué un paquete determinado no puede ser instalado en el proyecto actual, incluyendo cualquier conflicto de versiones o dependencias.
- **composer why <package>**: Este comando muestra qué paquetes hacen que se instale el paquete dado.
- **composer self-update**: Este comando actualiza la instalación de Composer a la última versión disponible.
- **composer outdated**: Este comando muestra las dependencias de un proyecto que están desactualizadas y las versiones más recientes disponibles.
- **composer create-project**: Este comando se utiliza para crear un proyecto a partir de un paquete Composer existente. Por ejemplo, se puede utilizar para instalar un framework como Laravel o Symfony.
- **composer global require <package>**: Este comando instala una dependencia globalmente en el sistema, lo que permite utilizarla en cualquier proyecto sin tener que instalarla manualmente.
- **composer prohibits <package>**: Este comando muestra qué paquetes impiden que se instale el paquete dado.
- **composer home <package>**: Este comando muestra la página del repositorio del paquete dado, si no se ofrece, en un proyecto Laravel por ejemplo se abrirá la página del repositorio de Laravel Framework.
- **composer suggests**: Este comando muestra sugerencias de paquetes para nuestro proyecto.
- **composer exec <binary>**: Este comando ejecuta un script binario del directorio vendor/bin.

- **composer fund**: Este comando muestra cómo ayudar a financiar el mantenimiento de las dependencias utilizadas.
- **composer licenses**: Este comando muestra información acerca del tipo de licencias de las dependencias utilizadas.
- **composer status**: Este comando muestra una lista de paquetes modificados localmente.