Readme

Indicaciones para un proyecto con código PHP y con implementación de COMPOSER, DOCTRINE y TWIG .

El paquete Twig es un motor de renderización de plantillas
desarrollado por el equipo de Symfony, aunque puede ser usado fuera de este Framework como es el caso.
En el siguiente enlace https://twig.symfony.com/doc/3.x/templates.html está toda la información respecto a estas plantillas como su estructura, uso o sintaxis entre otros.

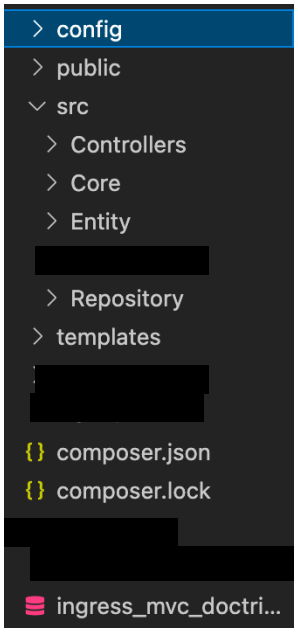
ORM (Object Relational Mapping), es una técnica de programación que convierte datos entre sistemas de tipos de un lenguaje de programación orientado a objetos y una base de datos relacional.

Requerimientos del sistema

Tener PHP instalado.
Otra opción es instalar XAMPP el cual incluye la instalación de MySQL, Apache y PHP

Estructura del proyecto

Crear un directorio con el cual trabajaremos para el proyecto.
Este directorio, mantiene una estructura típica de MVC.
En este proyecto tendremos los siguientes directorios.



Para la correcta ejecución del proyecto se necesitan las siguientes instalaciones:

Instalación de Composer

En la página oficial de [Composer](http://getcomposer.org) se encuentra toda la documentación necesaria y los pasos a seguir para la instalación.
En este caso, para MAC, en la terminal escribir los siguientes comandos:

- php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
- php -r "if (hash_file('sha384', 'composer-setup.php') === '906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cba5b96ac8b64115ac402c8cd292b8a03482574915d1a8') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

- php composer-setup.php

- php -r "unlink('composer-setup.php');"

En la ruta que hemos ejecutado los comandos, aparecerá un fichero composer.phar
Para instalarlo en global, movemos el fichero **composer.phar** a la carpeta /bin :
sudo mv composer.phar /usr/local/bin/composer

Para comprobar que se ha instalado correctamente escribimos en la terminal:
composer

Instalar Twig

En la terminal, situados en el mismo directorio que el fichero composer.json, en este caso en la raíz del proyecto, ejecutar **composer require "twig/twig**

Instalar Doctrine

En la terminal, situados en el mismo directorio que el fichero composer.json, en este caso en la raíz del proyecto, ejecutar:
composer require "doctrine/orm"
composer require "doctrine/cache"
composer require "symfony/cache"
composer require "doctrine/annotations"

Inicio proyecto

En la terminal instalamos composer con el comando **composer install** . Este paso creará una carpeta **vendor** en la cual se encuentra un fichero **autoload.php** y también creará un fichero **composer.lock**, todo en la raíz del proyecto.

Actualizar cambios de directorios

Mencionar que si se cambia la estructura de directorios ya sea añadiendo, quitando o moviendo de lugar, hay que ejecutar el comando **composer dump-autoload**. De lo contrario, composer no detectará los cambios y fallará.

Explicación del modelado de la BB.DD

Como sabemos, doctrine convierte una base de datos a objetos y viceversa.
Cada tabla, se corresponde con una de las clases definidas en el proyecto. Y cada campo de una tabla se corresponde con un atributo de una clase. Y estas clases, se disponen dentro del directorio Entity.

Todas las clases deben tener sus correspondientes comentarios para indicar a qué tabla hacen referencia. Los atributos también tienen sus comentarios sobre la columna o si indican una relación.

Todos los atributos tienen getter y setter para poder acceder a ellos desde fuera de la clase. Excepto el campo autonumérico que solo tiene getter.

Explicación del modelado

Dentro del directorio Core, creo la clase EntityManager.php la cual obtiene los datos para conectar a la bbdd a partir del fichero dbConfig.json situado en el directorio config. Además, en un atributo de la clase guardamos el EntityManager.

Como sabemos, doctrine convierte una base de datos a objetos y viceversa.
Cada tabla, se corresponde con una clase. Y cada campo de una tabla se corresponde con un atributo de la clase. Estas clases, se disponen dentro del directorio Entity. Todas las clases deben tener sus correspondientes comentarios para indicar a qué tabla hacen referencia. Los atributos también tienen sus comentarios sobre la columna o si indican una relación.

En el directorio repository creamos un repositorio por cada clase con el nombreClaseRepository.php. Todos los repositorios, extienden de EntityRepository el cual nos proporciona los métodos para hacer las consultas a la bbdd.

A partir del diagrama de tablas en phpmyadmin, podemos ver que las relaciones de las tablas son de 1 a N. En las clases 1 he creado un atributo con el nombre de la clase con la que se relaciona con su correspondiente comentario OneToMany. Y en el lado N de la relación también he creado un atributo con el nombre de la clase 1 y he añadido el comentario ManyToOne y Join ej:

```
use Doctrine\Common\Collections\ArrayCollection;

class Agent (lado 1)
/**
     * 1 agent tiene N Stats
     * @ORM\OneToMany(targetEntity="Stats", mappedBy="agent")
     */
    private $stats;

public function __construct() {
        $this->stats = new ArrayCollection()
}


class Stats (lado N)
/**
     * N Stats tiene 1 Agent
     * @ORM\ManyToOne(targetEntity="Agent", inversedBy="stats")
     * @ORM\JoinColumn(name="id_agent",
referencedColumnName="id_agent")
     */

    private $agent;
```

En cuanto a la lógica del proyecto, hay métodos tanto en los repositorios como en los controllers ya que en algunos casos hacen falta varias pequeñas consultas a diferentes tablas.

Respecto a la rúbrica de la entrega anterior, he indicado los campos que pueden ser null así como los que son únicos. También he cambiado el tipo de dato de id_event en la tabla uploads por un boolean.

En cuanto a la relación de OneToOne de Stats y StatsEvent con Upload, en mi caso sigue siendo OneToMany porque al intentar cambiar a OneToOne el código ha empezado a fallar con errores que desconozco y también por falta de tiempo prefiero entregar el código funcionando.

También he intentado gestionar errores mostrando diferentes mensajes y ocultando contenido según el caso.

Por último mencionar, que al final del vídeo el resultado de la comparativa del 1er evento, varios campos dan negativo porque en la última estadística de prueba subida, son null, ahí la causa.