

# Doctrine

Version 1.2.0

Niveau requis : 4/7



## *Mise en œuvre de Doctrine 2.6 (O.R.M. PHP)*

## Sommaire

<b>I.</b>	<b>PREAMBULE.....</b>	<b>3</b>
I.I.	OBJET.....	3
I.II.	PRE-REQUIS.....	3
I.III.	VERSIONS DU DOCUMENT.....	3
I.IV.	DOCUMENTS DE REFERENCE.....	3
<b>II.</b>	<b>QU'EST CE QU'UN ORM ? .....</b>	<b>4</b>
II.I.	DESCRIPTION.....	4
II.II.	EXEMPLE PERSONNE - COURS.....	4
<b>III.</b>	<b>INSTALLATION DE DOCTRINE.....</b>	<b>5</b>
III.I.	CREATION DU PROJET « COURS ».....	5
III.II.	CREATION DU VIRUTAL HOST « COURS ».....	7
III.III.	INSTALLATION DE LA LIBRAIRIE DOCTRINE.....	8
III.III.1	<i>Ajout de la prise en charge de Composer</i> .....	8
III.III.2	<i>Récupération de Doctrine</i> .....	10
III.IV.	POSITIONNEMENT DE L'ENVIRONNEMENT « DEVELOPMENT ».....	13
III.V.	CONFIGURATION DE DOCTRINE.....	15
III.V.1	<i>Configuration d'accès à la base de données</i> .....	15
III.V.2	<i>Création du bootstrap.php</i> .....	16
<b>IV.</b>	<b>CREATION DU SCHEMA DE LA BASE .....</b>	<b>19</b>
IV.I.	CREATION DU SCHEMA COURS DANS LA BASE DE DONNEES MYSQL.....	19
IV.II.	CREATION DU FICHIER CLI-CONFIG.PHP.....	21
IV.III.	LANCEMENT DANS LA CONSOLE MSDOS.....	22
IV.IV.	CREATION DES ENTITY SANS METTRE LES RELATIONS.....	23
IV.V.	VERIFIONS CE QUI S'EST PASSE EN BASE DE DONNEES.....	26
IV.VI.	MISE EN PLACE DES RELATIONS.....	27
IV.VI.1	<i>Relation ManyToOne</i> .....	27
IV.VI.2	<i>Relation inverse OneToMany</i> .....	29
IV.VI.3	<i>Relation ManyToMany</i> .....	32
<b>V.</b>	<b>MISE EN ŒUVRE DE L'ACCES A LA BASE DE DONNEES.....</b>	<b>36</b>
V.I.	OBJECTIF.....	36
V.II.	AJOUT DES « REPOSITORY ».....	36
V.III.	MISE A JOUR DES ENTITY POUR RAJOUTER LA DECLARATION DES REPOSITORY.....	39
V.IV.	MISE EN PLACE DE LA COUCHE SERVICE.....	40
V.V.	AFFICHAGE DES DONNEES.....	44
<b>VI.</b>	<b>SOURCES D'INFORMATIONS.....</b>	<b>46</b>
<b>VII.</b>	<b>FIN DU DOCUMENT .....</b>	<b>46</b>

## I. Préambule

### I.I. *Objet*

L'objet de ce document est de présenter l'approche de mise en œuvre de l'Object Relational Mapper (O.R.M.) Doctrine 2.6 en PHP.

### I.II. *Pré-requis*

Avoir un environnement de développement installé avec PHP7, base de données MySQL 5.

Nous allons partir du principe que vous avez suivi les documents suivants :

1. Installation Wamp\_1.0.2
2. Installation Zend Studio\_1.0.1
3. Debugger en PHP\_1.0.1

Dans cette formation, nous utiliserons Zend Studio 13.6

Avoir téléchargé HeidiSQL : sur <https://www.heidisql.com/>

### I.III. *Versions du document*

Version	Date	Auteur	Description
1.0.0	24/03/2018	Péquignat.eu	Version initiale du document
1.0.1	27/03/2018	Péquignat.eu	Mise en place de l'accès à la base de données
1.2.0	05/03/2022	Péquignat.eu	Retrait de l'autoentreprise

### I.IV. *Documents de référence*

#	Document	Version	Auteur(s)
[R1]	Installation Wamp	1.0.2	Péquignat.eu
[R2]	Installation Zend Studio	1.0.1	Péquignat.eu
[R3]	Debugger en PHP	1.0.1	Péquignat.eu
[R4]	Virtual Host Wamp	1.0.1	Péquignat.eu

## II. Qu'est ce qu'un ORM ?

### II.I. Description

L'Object Relational Mapping est une petite architecture sous forme d'API permettant de faire le lien entre le modèle Objet représentant ses entités de travail et la base de données. Ce moteur permet de travailler sous forme d'une représentation Objet avec ses différentes relations entre Objets :

- 1..1 : One to One
- 1..N : Many to One
- N..1 : One to Many
- N..N : Many To Many

Avec les objets et ses relations, l'ORM fait l'interface avec la base de données (Ex : MySQL) pour assurer la persistance.

### II.II. Exemple Personne - Cours

Nous allons présenter un exemple de mise en place d'un modèle objet pour ensuite avoir sa représentation en base de données.

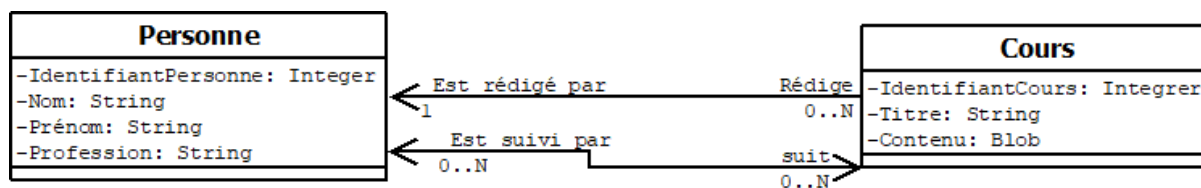


Figure 1 - Personne - Cours

Nous avons ici deux entités : une Entité Personne et une Entité Cours.

Entre ces entités, nous avons plusieurs relations :

- ⇒ Cours qui est rédigé par une et une seule Personne : **OneToMany**
- ⇒ Relation Personne qui rédige zéro ou plusieurs Cours : **ManyToOne**
- ⇒ Un Cours qui peut être suivi par plusieurs Personnes et une Personne qui peut suivre plusieurs Cours : **ManyToMany**

La mise en œuvre en Modèle de table est la suivante :

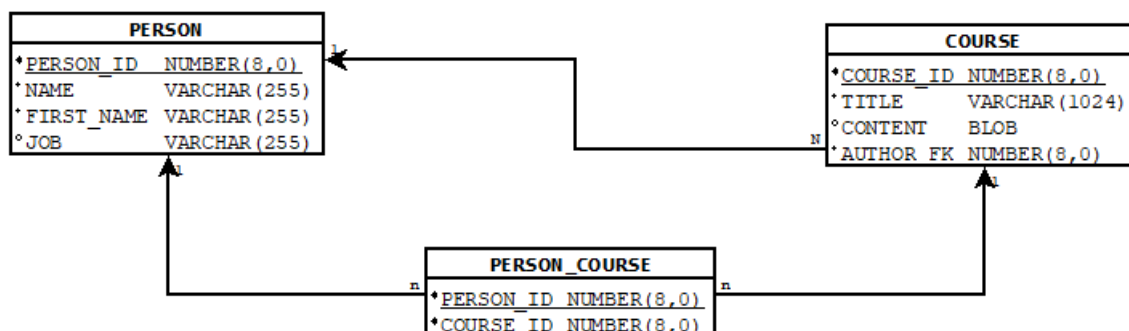


Figure 2 - Modèle de base de données

Nous avons ici trois tables, car en effet nous avons la relation ManyToMany qui est représentée par une table de lien PERSON\_COURSE.

La relation ManyToOne d'une Personne vers le Cours et donc le OneToMany du Cours vers la Personne est représentée par une clé étrangère AUTHOR\_FK dans la table COURSE qui fait référence à PERSON\_ID de la table PERSON.

### III. Installation de Doctrine

#### III.I. Création du Projet « Cours »

Créer un projet « PHP Local Project » Cours comme suit :

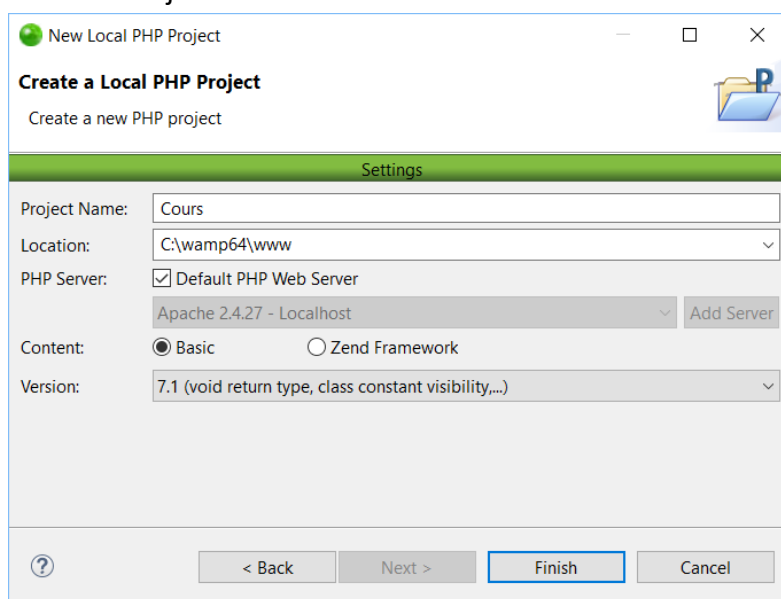


Figure 3 - Création Projet Cours

Le projet s'est créé :

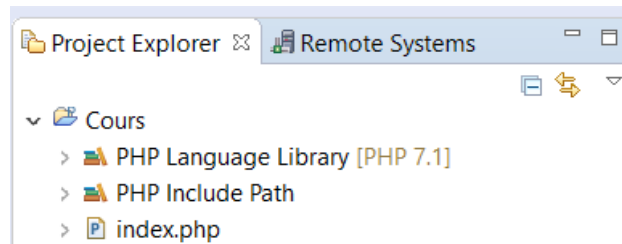


Figure 4 - Projet Cours créé

Cette répartition du fichier index.php à la racine du répertoire cours n'est pas optimale. En effet, nous allons créer des sous répertoire à Cours comme « vendor », et une mauvaise utilisation pourrait laisser des failles.

Aussi nous allons créer un sous répertoire de « Cours » nommé « web ».

Nous y déplaçons le fichier index.php dans ce répertoire « web ».

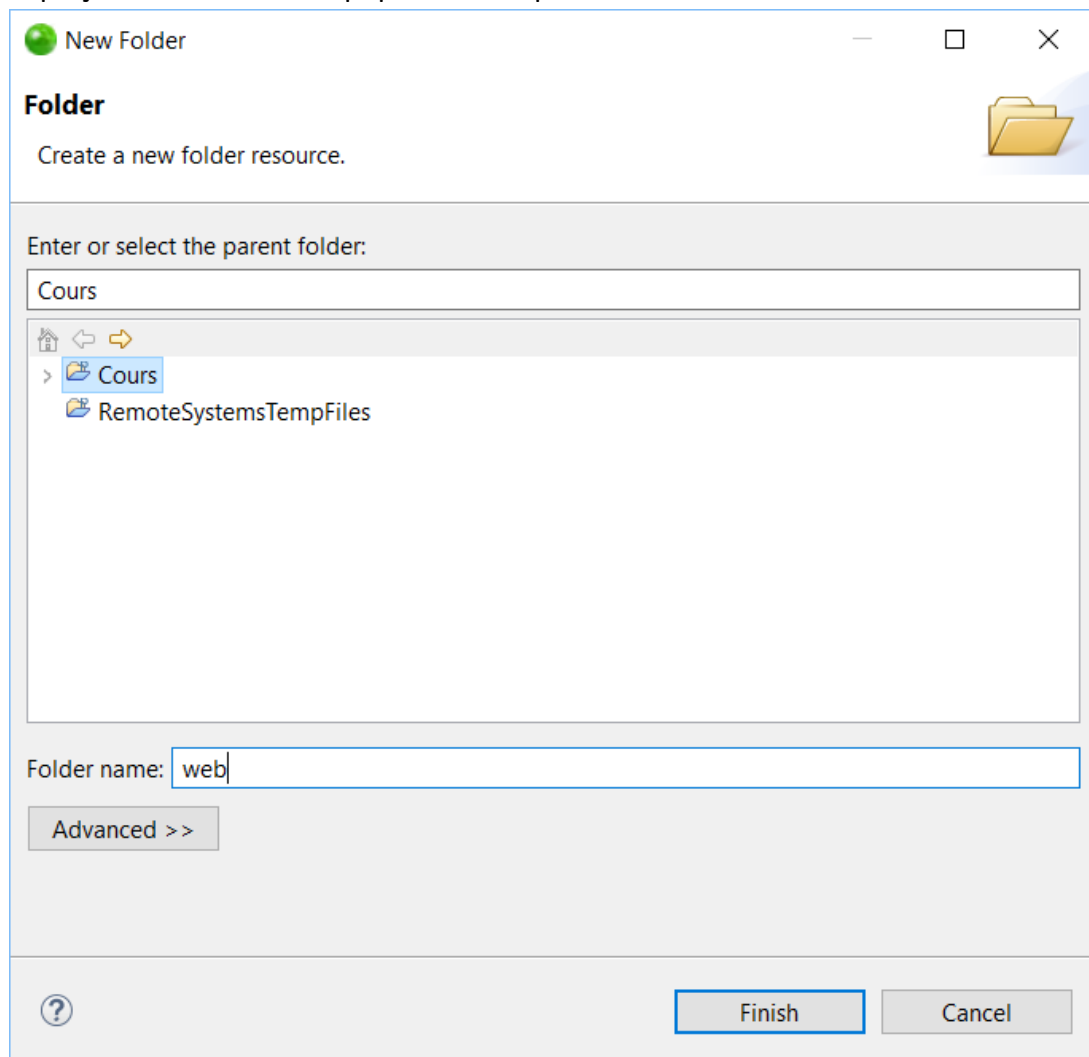


Figure 5 - Création du répertoire web

Cliquez sur Finish

Faite ensuite un déplacement du fichier index.php dans ce nouveau répertoire.

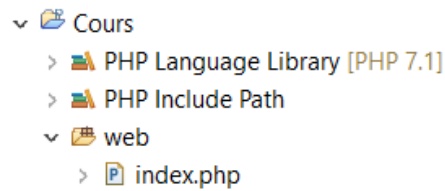


Figure 6 - Déplacement index.php

### III.II. Création du Virtual Host « Cours »

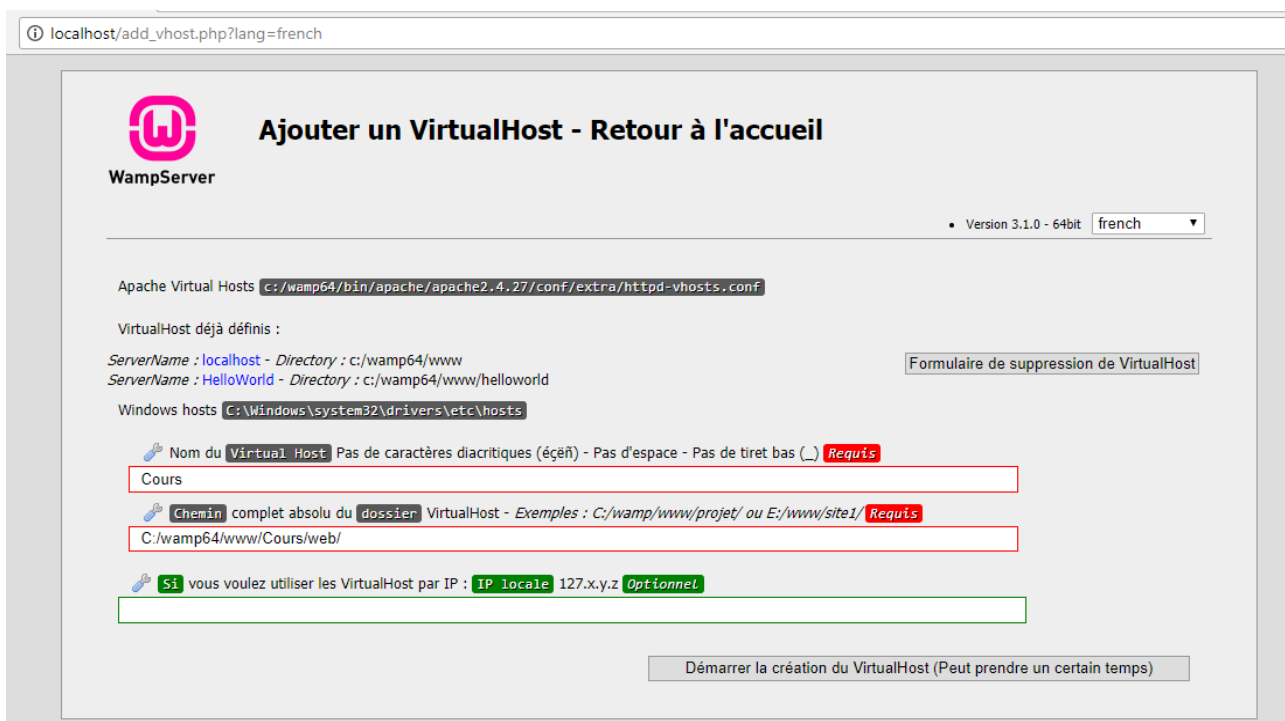
Allez dans <http://localhost>

Cliquez sur Ajouter un Virtual Host :

Nom : Cours

Chemin : C:/wamp64/www/Cours/web/

Attention, mettre volontairement le sous répertoire « web » comme point d'entrée.



localhost/add\_vhost.php?lang=french

**WampServer**

Ajouter un VirtualHost - Retour à l'accueil

Version 3.1.0 - 64bit french

Apache Virtual Hosts `c:/wamp64/bin/apache/apache2.4.27/conf/extra/httpd-vhosts.conf`

VirtualHost déjà définis :

ServerName : localhost - Directory : c:/wamp64/www

ServerName : HelloWorld - Directory : c:/wamp64/www/helloworld

Windows hosts `C:\Windows\system32\drivers\etc\hosts`

Nom du Virtual Host `Pas de caractères diacritiques (éçëñ) - Pas d'espace - Pas de tiret bas (_) Requis`

Cours

Chemin complet absolu du dossier VirtualHost - Exemples : C:/wamp/www/projet/ ou E:/www/site1/ Requis

C:/wamp64/www/Cours/web/

Si vous voulez utiliser les VirtualHost par IP : IP locale 127.x.y.z Optionnel

Démarrer la création du VirtualHost (Peut prendre un certain temps)

Figure 7 - Ajout du Virtual Host Cours

Cliquez sur « Démarrer la création du Virtual Host »

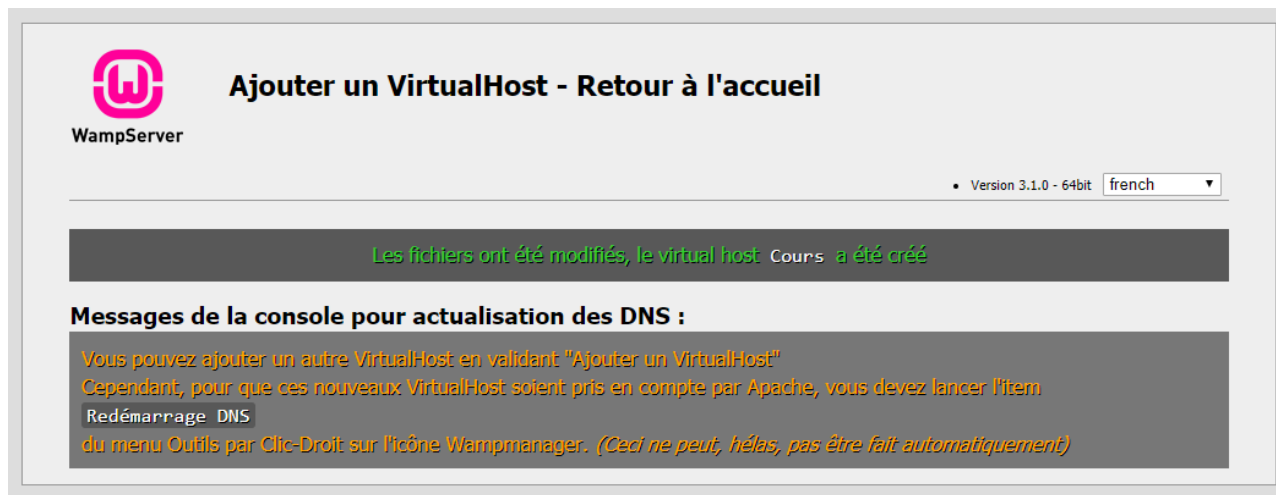


Figure 8 - Création réussi du VH

Redémarrer les services.

Vous pouvez cliquer sur <http://cours>

Vous devriez avoir une page blanche.

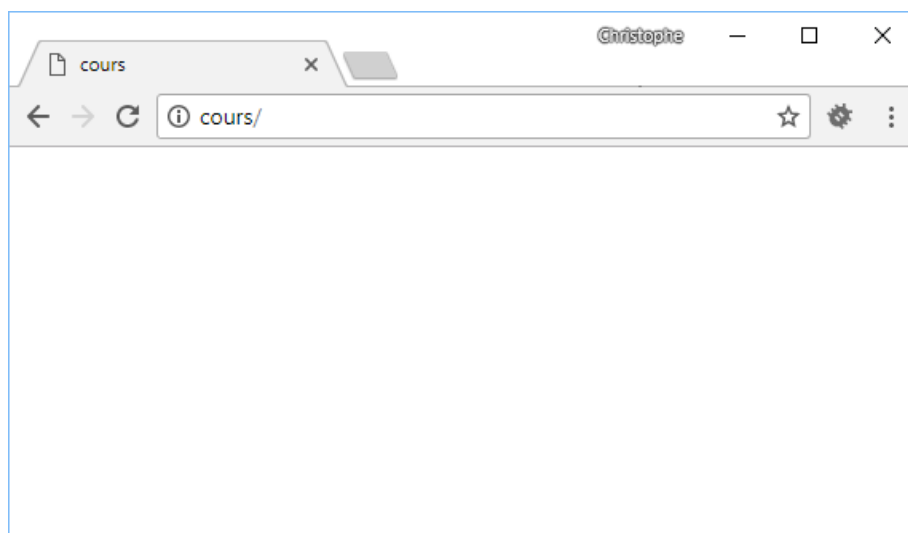


Figure 9 ( Page Cours

### III.III. Installation de la librairie DocTrine

#### III.III.1 Ajout de la prise en charge de Composer

Nous allons utiliser « Composer » fourni avec Zend Studio afin de récupérer dans son projet : Doctrine en version 2.6.

Pour cela, faites un clique droit sur la souris sur le projet Cours, puis aller dans « Configure » et enfin cliquer sur « Add Composer Support »



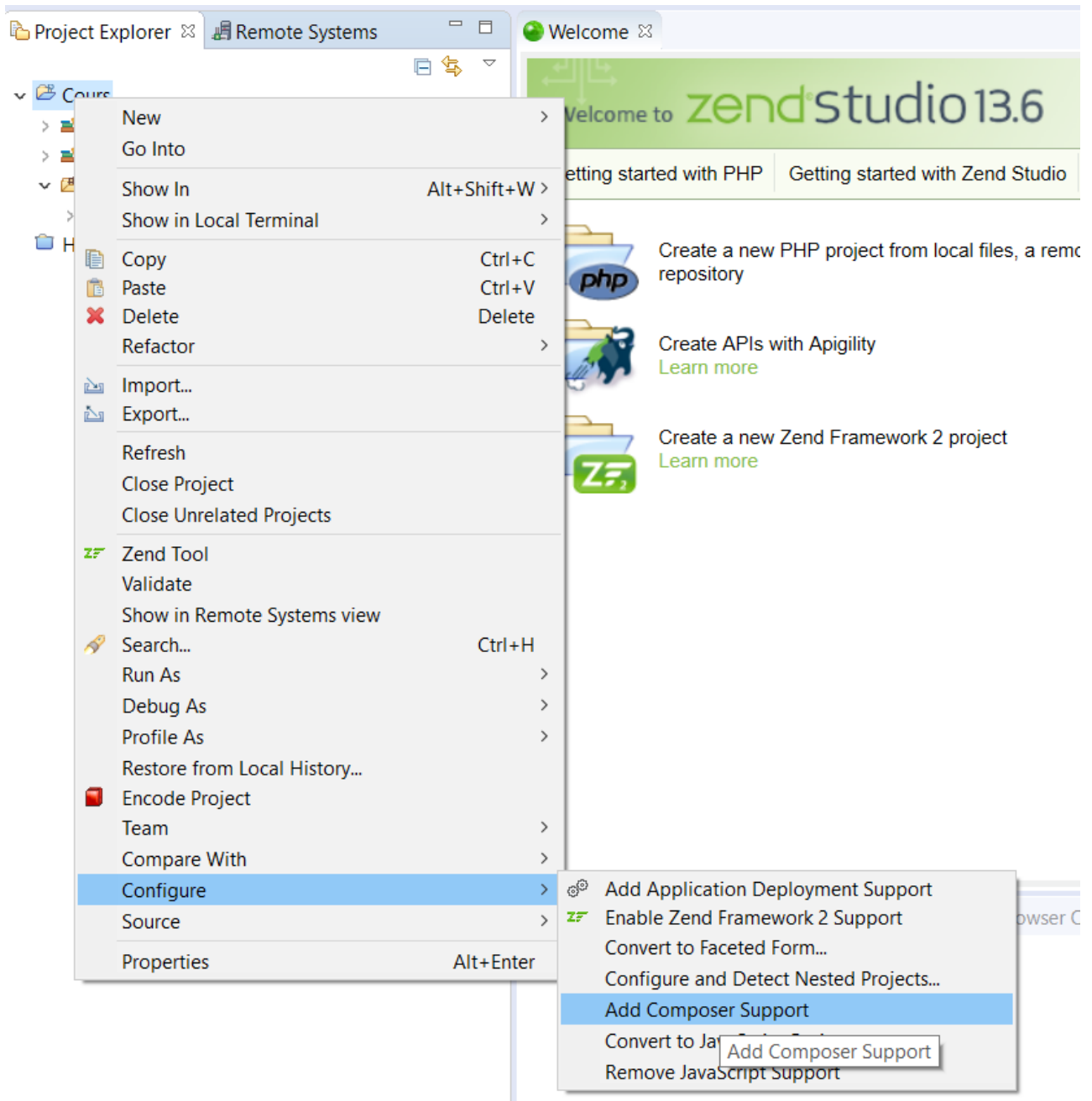


Figure 10 - Ajout de la prise en charge de Composer

Un nouveau fichier s'est créé : « composer.json » à la racine du projet « Cours ».  
Ce fichier sert à gérer le projet et les dépendances.

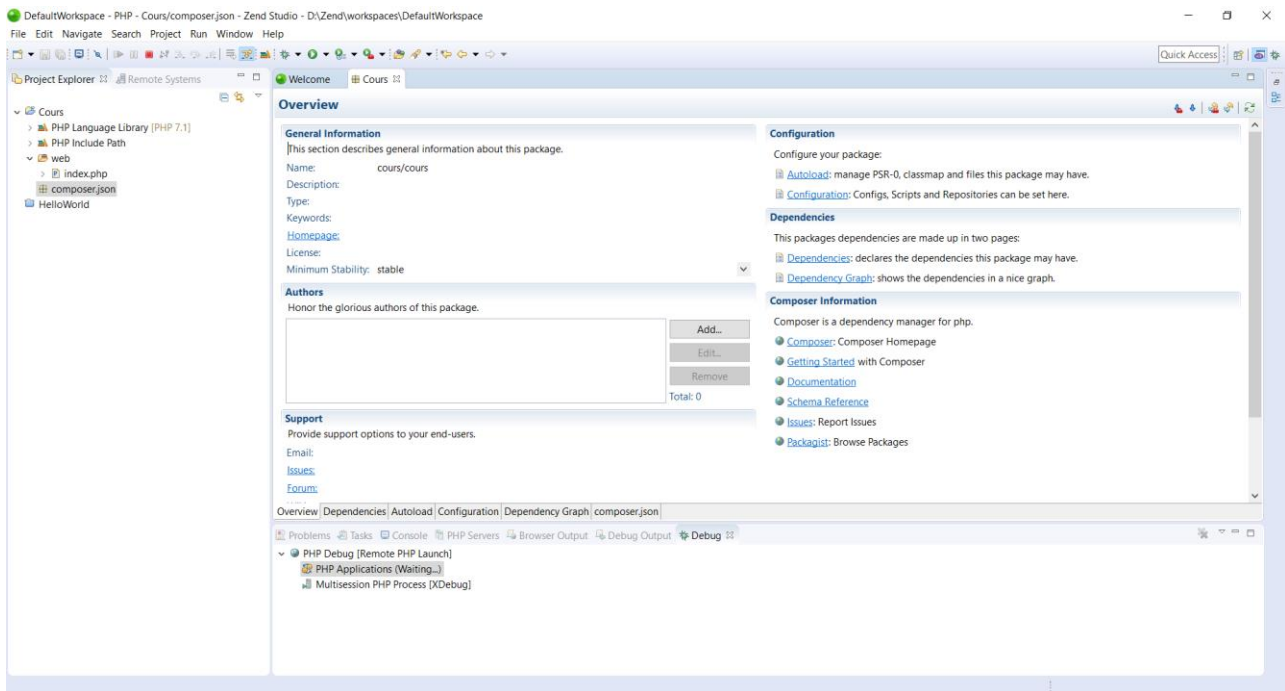


Figure 11 - composer.json

### III.III.2 Récupération de Doctrine

Allez dans la vue « composer.json » et rajouter les informations suivantes :

On va renommer le projet en « pequignat.eu/cours »

Dans la partie « require » : c'est la que nous demandons la dernière version 2.6.\* existante

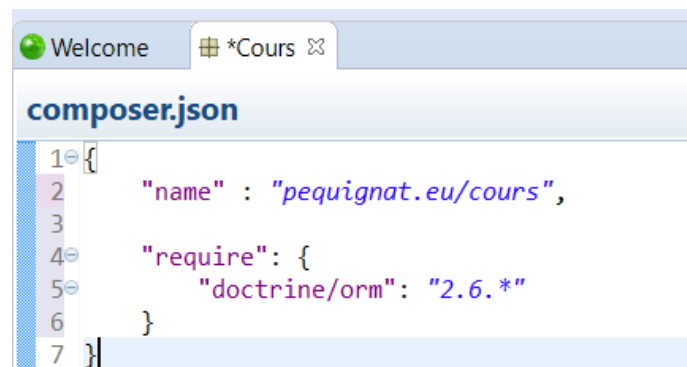


Figure 12 - configuration de Doctrine

```
{  
  
    "name" : "pequignat.eu/cours",  
  
    "require": {  
        "doctrine/orm": "2.6.*"  
    }  
}
```

Faite un Ctrl+S pour sauvegarder les modifications.

Afficher ensuite la « View » Console comme suit :

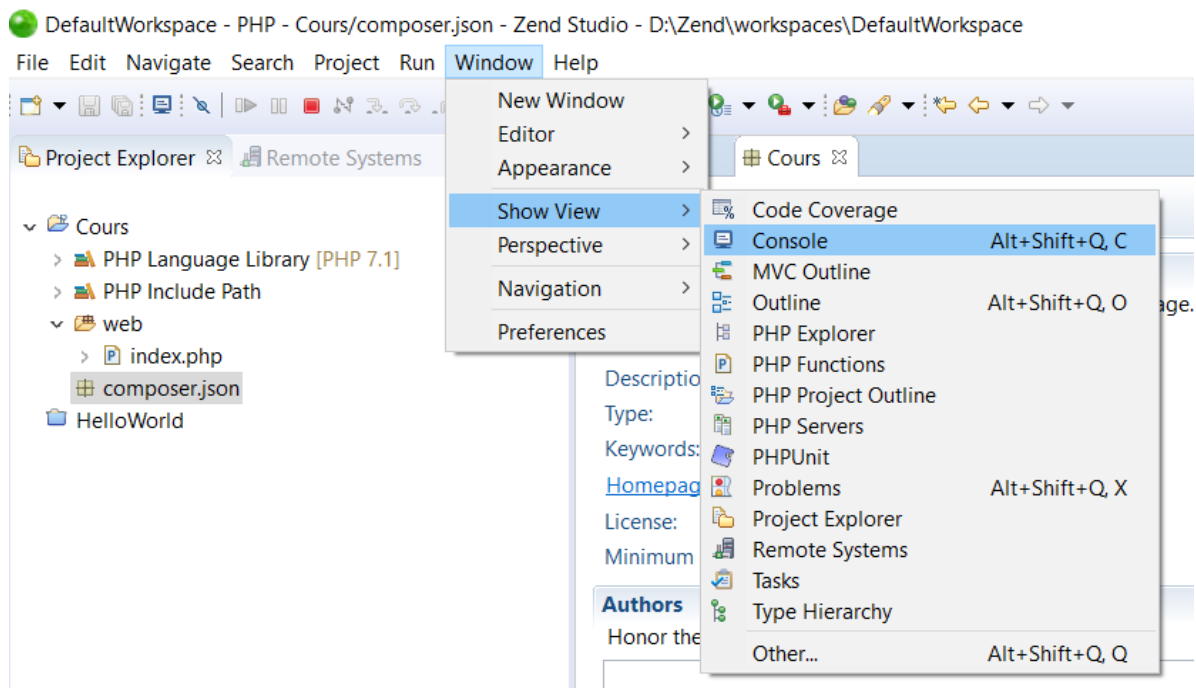


Figure 13 - Ajout de la Console

Lancer un « Update Dependencies » sur la dépendance doctrine/orm:2.6.\*

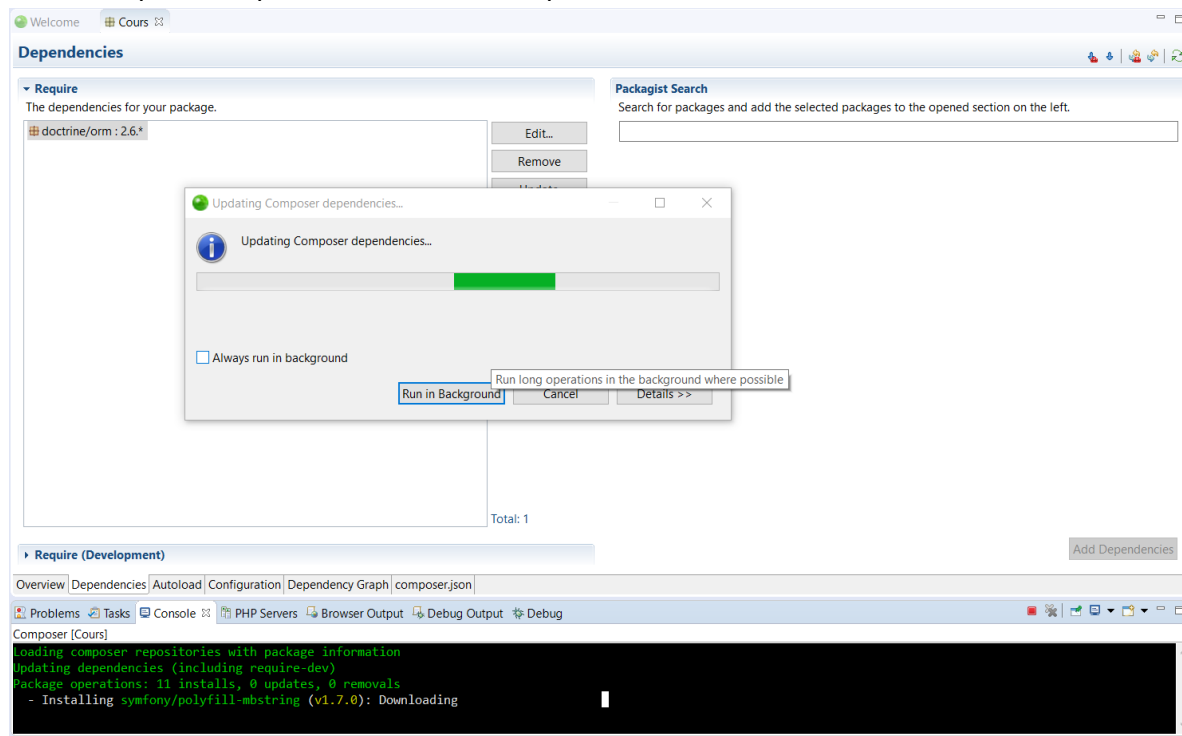


Figure 14 - Update Dependencies

Cela a dû créer le répertoire « vendor » contenant :

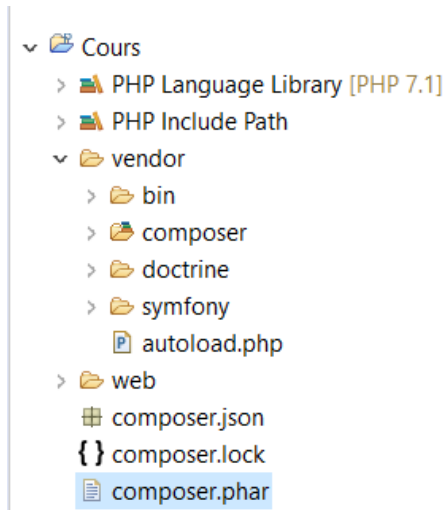


Figure 15 - Vendor

Maintenant rajouter dans le composer.json :

```
{
    "name" : "pequignat.eu/cours",
    "require" : {
        "doctrine/orm" : "2.6.*"
    },
    "autoload" : {
        "psr-0" : { "" : ["config/", "src/"] }
    }
}
```

Faites un « Update dependencies »

Et créer les répertoires à la racine de Cours : config et src

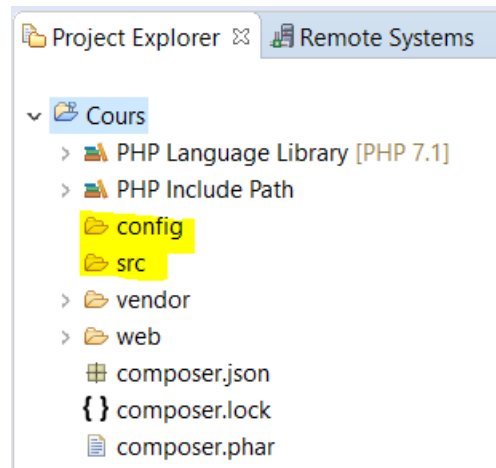


Figure 16 - Répertoire src et config

### III.IV. *Positionnement de l'environnement « development »*

Dans le répertoire Cours/web créé un fichier nommé « .htaccess ».

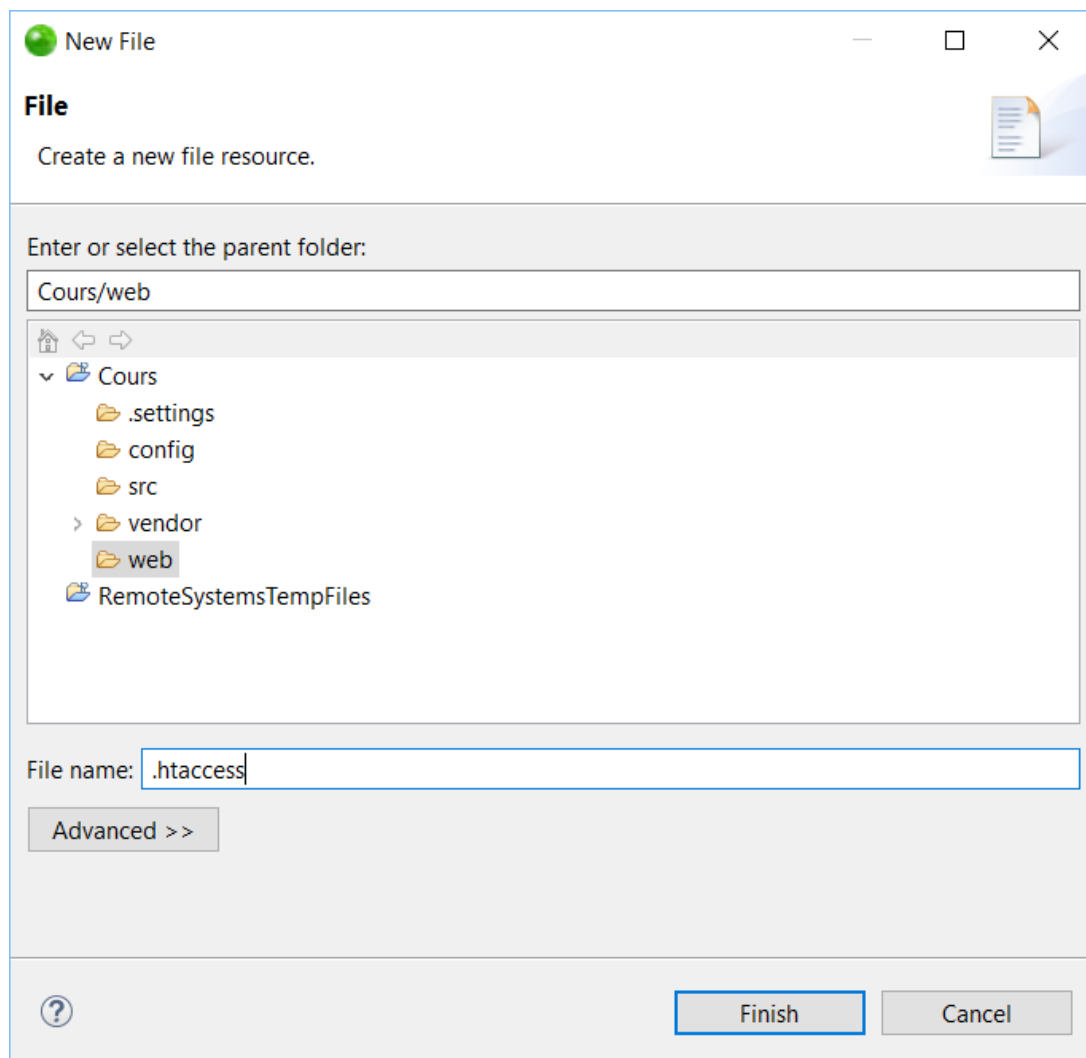


Figure 17 - création du .htaccess

Activer la vue « Navigator » dans « Windows » → « Show View »

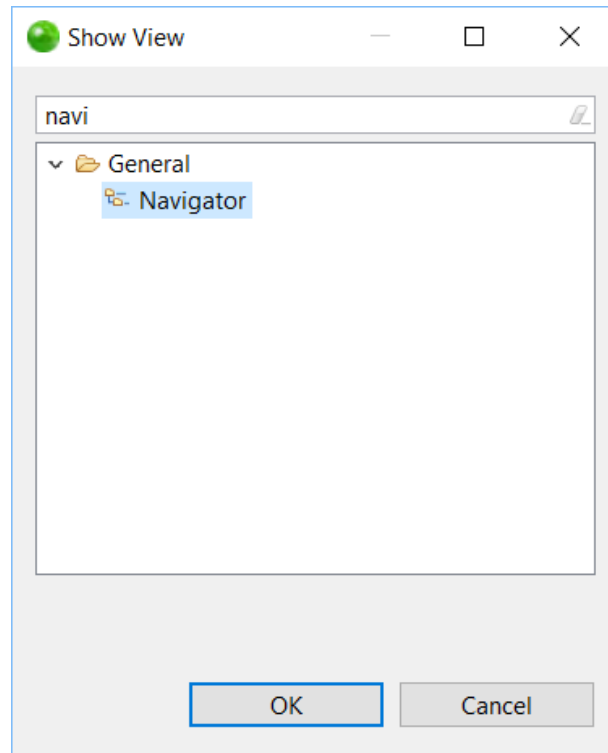


Figure 18 - Vue Navigator

Cliquer sur OK

Le fichier « .htaccess » est maintenant visible.

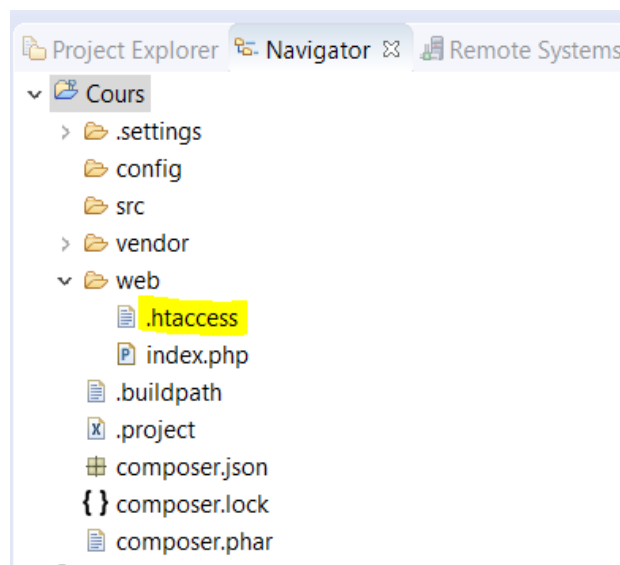


Figure 19 - Navigator

Allez dans le fichier « .htaccess » et rajouter la ligne :

```
SetEnv APPLICATION_ENV development
```

On se servira de « APPLICATION\_ENV » pour déterminer sur quel environnement on se situe.

Valeurs préconisées : « development » ou « production ».

Rajouter maintenant dans l'index.php de web le contenu suivant :

```
<?php
defined ("APPLICATION_ENV") || define("APPLICATION_ENV",
getenv('APPLICATION_ENV') ?? 'production');
echo APPLICATION_ENV;
```

Et afficher la page : <http://cours>

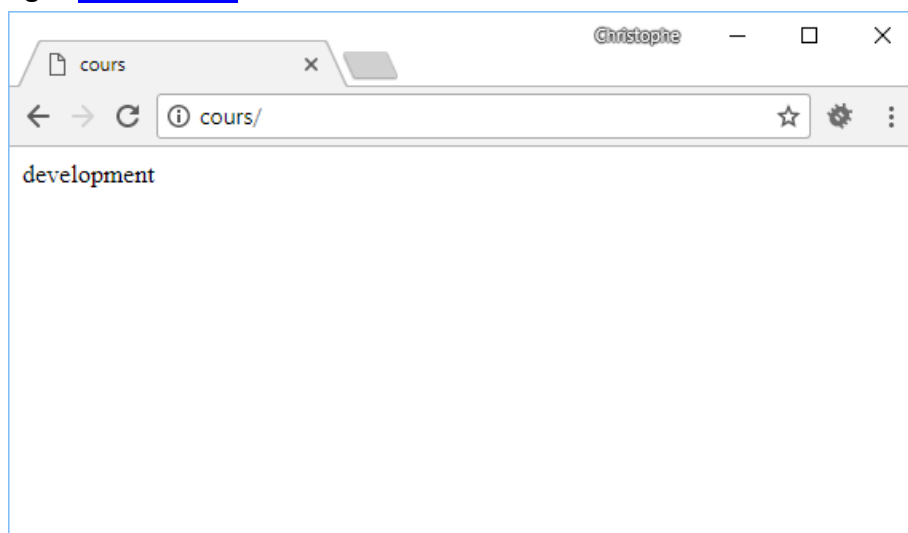


Figure 20 - affichage de l'environnement

Attention : chez un Hébergeur tel qu'OVH, le passage de variable d'environnement est désactivé sur l'hébergement mutualisé.

Il convient dans l'utilisation de définir une valeur par défaut étant la « production » dans le code PHP.

### III.V. Configuration de Doctrine

#### III.V.1 Configuration d'accès à la base de données

Créons un fichier config/Database.php contenant les accès à la base de données MySQL.

```
<?php
class Database
{
    /**
```

```
* Renvoie les paramètres de connexion
* @return array
*/
public static function getConnectionParams() : array
{
    return array(
        'dbname' => 'cours', // Le nom du schema de la base
        'user' => 'root', // L'utilisateur de connexion
        'password' => '', // Le mot de passe
        'host' => 'localhost', // Le serveur ici localhost
        'driver' => 'pdo_mysql' // Le driver de connexion PDO
    );
}
}
```

Créons aussi un fichier « .htaccess » présent dans ce même répertoire pour le protégé :  
Avec le contenu :

```
deny from all
```

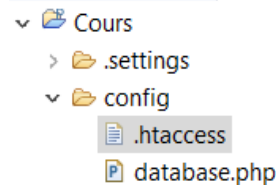


Figure 21 - Config

### III.V.2 Création du bootstrap.php

La classe Bootstrap.php est un fichier qui initialise toute l'application. Nous allons la créer dans le répertoire « src ».

Nous y allons y déclarer l'initialisation de la base de données. C'est là aussi que l'on peut initialiser la session PHP pour les sites ayant une session ou encore les Log avec Log4PHP...

```
<?php
// src/Bootstrap.php
use Doctrine\ORM\Tools\Setup;
```



```
use Doctrine\ORM\EntityManager;

class Bootstrap{

    /**
     *
     * @var bool
     */
    private static $isDevMode = false;

    /**
     *
     * @var EntityManager
     */
    private static $entityManager = null;

    private static function initDevMode() : void {
        self::$isDevMode = in_array( APPLICATION_ENV , array('development'));
    }

    private static function initDatabase() : void{
        // Create a simple "default" Doctrine ORM configuration for Annotations
        $config = Setup::createAnnotationMetadataConfiguration(array(__DIR__),
self::$isDevMode);
        // or if you prefer XML
        //$config =
Setup::createXMLMetadataConfiguration(array(__DIR__."/config"), $isDevMode);
        // database configuration parameters
        $conn =
\Doctrine\DBAL\DriverManager::getConnection(Database::getConnectionParams(),
$config);

        // obtaining the entity manager
        self::$entityManager = EntityManager::create($conn, $config);
    }
}
```

```
public static function init() : void{
    self::initDevMode();
    self::initDatabase();
}

/**
 * Donne access à l'EntityManager d'accès à la base
 * @return EntityManager
 */
public static function getEntityManager() : EntityManager{
    return self::$entityManager;
}
}
```

La méthode Bootstrap::init() initialise tout ce qui faut pour le démarrage.

Voyons maintenant le fichier bootstrap.php se situant à la racine du projet

```
<?php
// bootstrap.php
require_once "vendor/autoload.php";
Bootstrap::init();
```

Revenons au fichier index.php présent dans web...

Mettre le contenu suivant :

```
<?php
defined ("APPLICATION_ENV") || define("APPLICATION_ENV",
getenv('APPLICATION_ENV') ?? 'production');
echo APPLICATION_ENV . "<br/>\r\n";

echo 'Initialisation de la connexion à la base de données';
require_once ('..\bootstrap.php');
```



Figure 22 - Affichage de la connexion

## IV. Création du schéma de la base

### IV.I. Création du Schéma cours dans la base de données MySQL

Connectez-vous sur le lien : <http://localhost/phpmyadmin/>

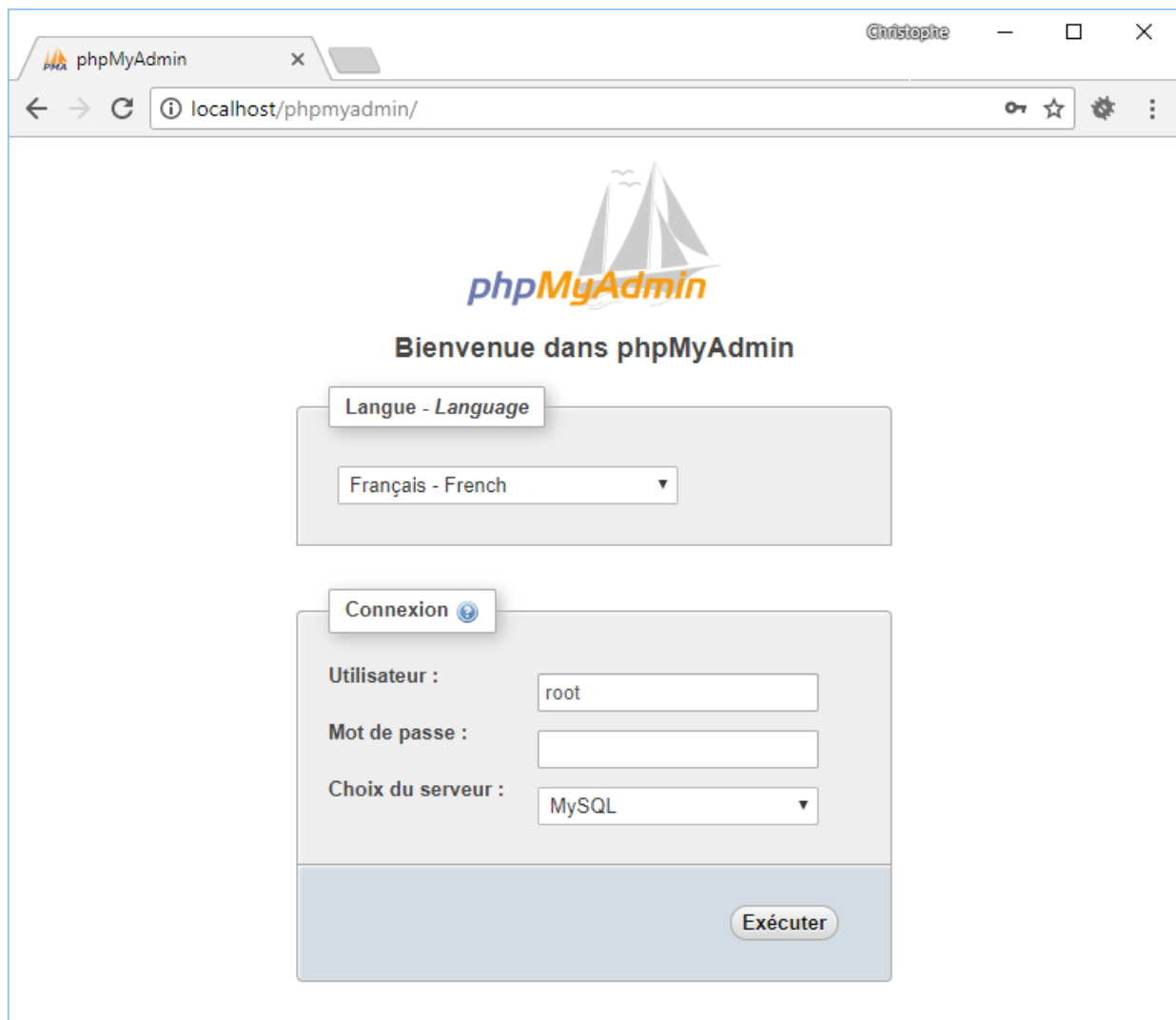


Figure 23 - Connexion PHPMyAdmin

Cliquez sur Exécuter

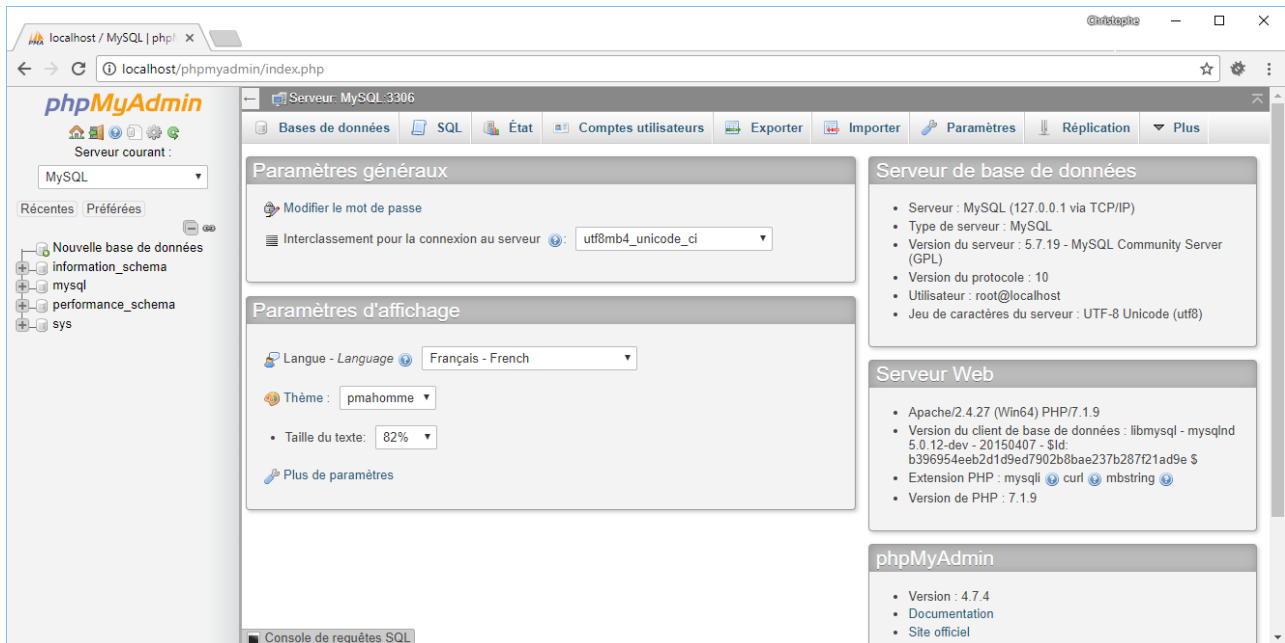


Figure 24 - Accueil PHPMYAdmin

Cliquez sur Bases de données

Compléter le nom de la base de données : cours

Encodage : utf8\_general\_ci



Figure 25 - Création du schéma de données

Cliquez sur « Créer »

On est redirigé vers :

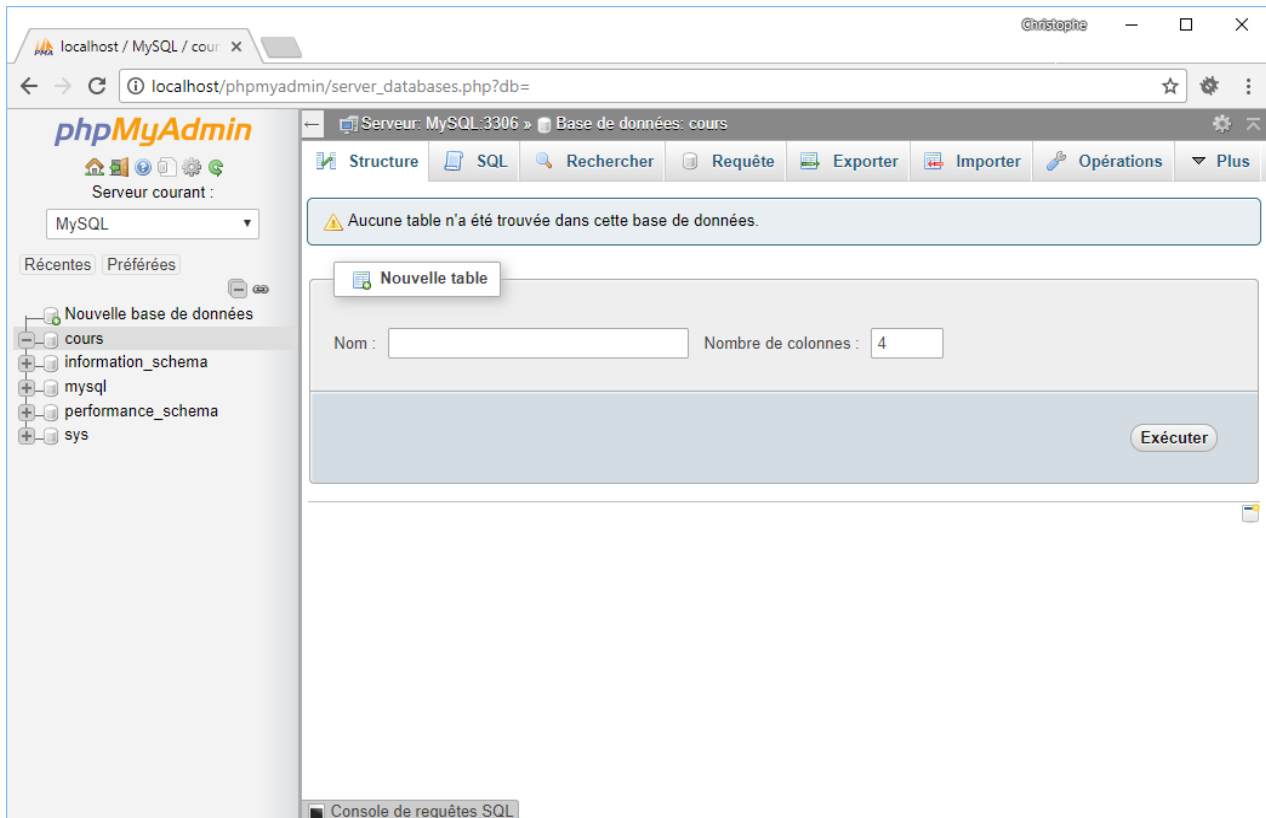


Figure 26 - Schéma cours créé

#### IV.II. Création du fichier `cli-config.php`

Doctrine a un applicatif en ligne de commande qui permet en outre de générer le schéma de la base de données Relationnel.

Cette génération se base sur les classe Entity de votre projet en utilisant les métadatas (Annotation dans les commentaires).

Créez le fichier « `cli-config.php` » à la racine de Cours

```
<?php
// cli-config.php
defined ("APPLICATION_ENV") || define("APPLICATION_ENV", 'development');
require_once "bootstrap.php";
return
\Doctrine\ORM\Tools\Console\ConsoleRunner::createHelperSet(Bootstrap::getEntityManager());
```

Nous allons initialiser le `bootstrap.php` et récupérer l'EntityManager pour le passer à la console.

### IV.III. *Lancement dans la console MSDOS*

Ouvrir une Commande « CMD » dans la barre de recherche de Windows.

Tapez :

```
cd /D c:\wamp64\www\Cours
```

Vous devriez voir ceci :

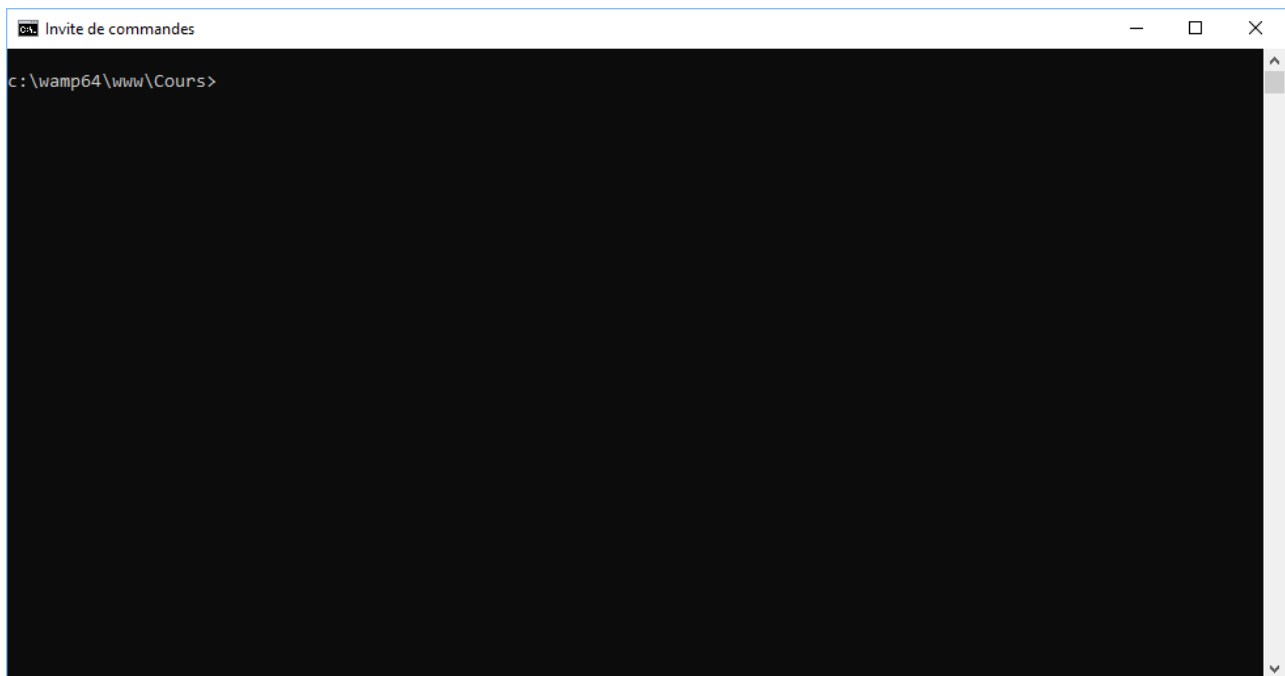
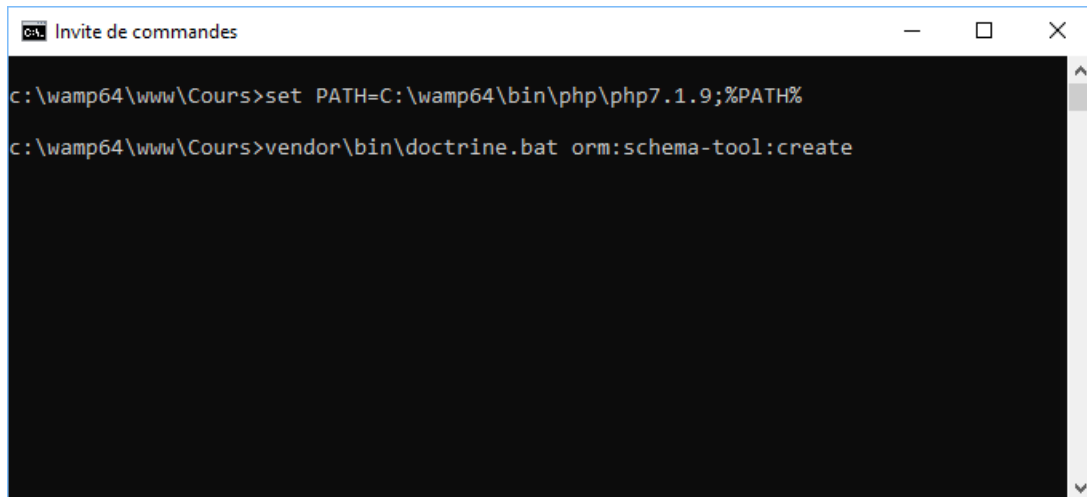


Figure 27 - Commande MSDOS

Dans la console tapez maintenant :

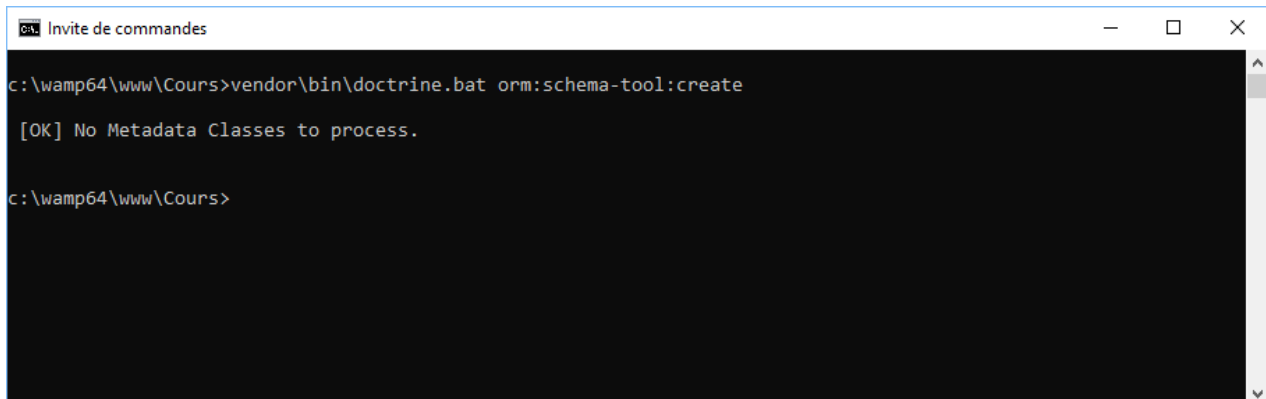
```
set PATH=C:\wamp64\bin\php\php7.1.9;%PATH%
vendor\bin\doctrine.bat orm:schema-tool:create
```



```
Invite de commandes
c:\wamp64\www\Cours>set PATH=C:\wamp64\bin\php\php7.1.9;%PATH%
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create
```

Figure 28 - Lancement commande création schéma

Le résultat de la commande si tout est bien initialisé :



```
Invite de commandes
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create
[OK] No Metadata Classes to process.
c:\wamp64\www\Cours>
```

Figure 29 - Fin du processus de création des tables

Nous verrons après la création des Entity Person et Course avec leur méta données.

Une fois mise en place, deux méthodes sont possibles pour mettre à jour le modèle de la base :

```
vendor\bin\doctrine.bat orm:schema-tool:drop --force
vendor\bin\doctrine.bat orm:schema-tool:create
```

ou

```
vendor\bin\doctrine.bat orm:schema-tool:update --force
```

#### IV.IV. *Création des Entity sans mettre les relations*

Créer dans le répertoire « src », un sous répertoire « Entity » et y créer les deux fichiers PHP :

- Person.php

- Course.php

Voici le contenu de ces Entity :

Pour Person.php

```
<?php
namespace Entity;

/**
 * @Entity
 * @Table(name="PERSON")
 * @author Christophe PEQUIGNAT
 */
class Person
{
    /**
     * @Id
     * @Column(type="integer", name="PERSON_ID")
     * @GeneratedValue
     * @var int
     */
    private $id;

    /**
     * @Column(type="string", length=255, name="NAME")
     * @var string
     */
    private $name;

    /** @Column(type="string", length=255, name="FIRST_NAME")
     * @var string
     */
    private $firstname;
```



```
/** @Column(type="string", length=255, name="JOB")
 * @var string
 */
private $job;
}
```

## Pour Course.php

```
<?php
namespace Entity;

/**
 * @Entity
 * @Table(name="COURSE")
 * @author Christophe PEQUIGNAT
 *
 */
class Course
{

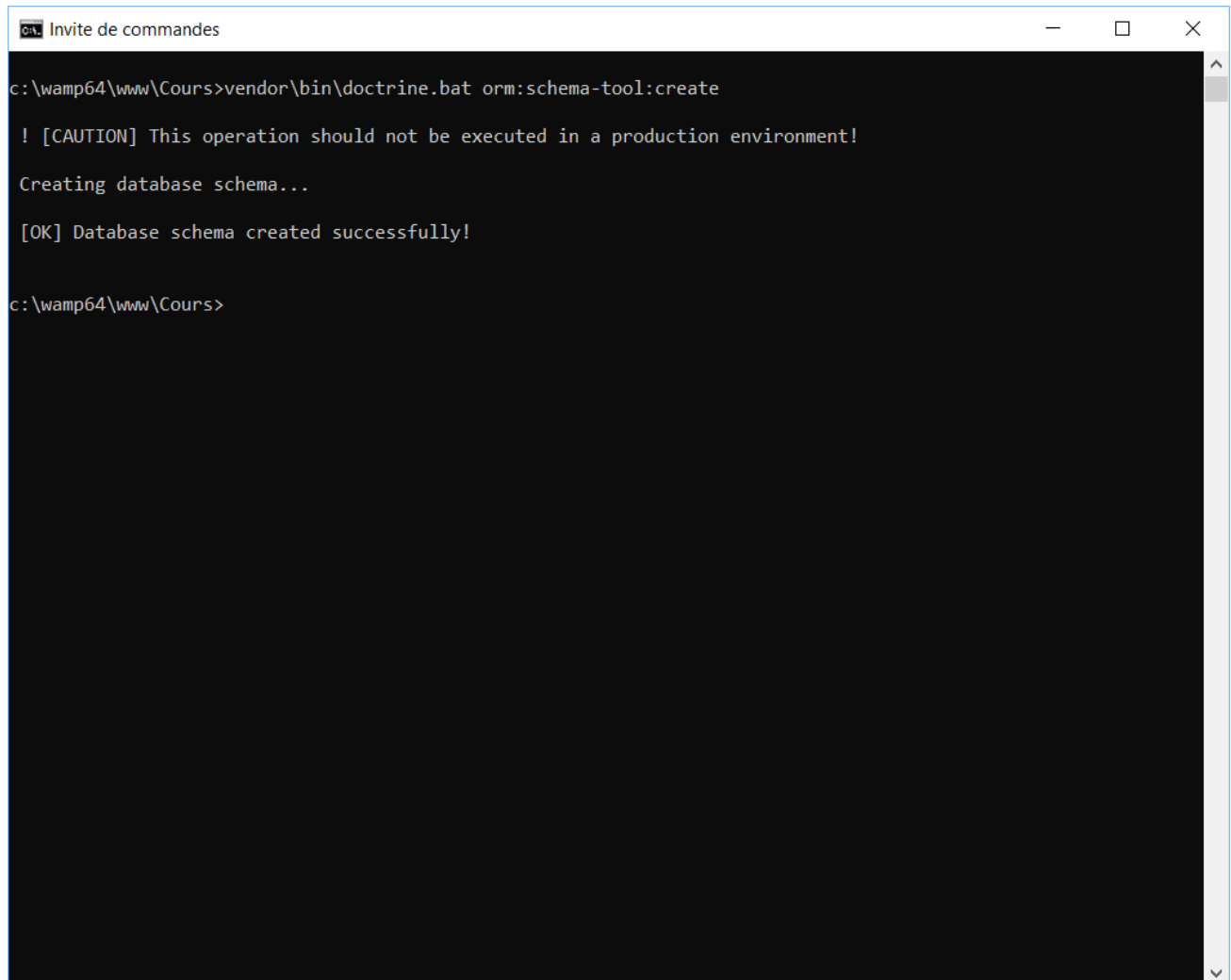
    /**
     * @Id
     * @Column(type="integer", name="COURSE")
     * @GeneratedValue
     * @var int
     */
    private $id;

    /**
     * @Column(type="string", length=1024, name="TITLE")
     * @var string
     */
    private $title;
```

}

Relançons la commande de mise à jour de la base de données :

```
vendor\bin\doctrine.bat orm:schema-tool:create
```



```
Invite de commandes
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create
! [CAUTION] This operation should not be executed in a production environment!
Creating database schema...
[OK] Database schema created successfully!
c:\wamp64\www\Cours>
```

Figure 30 - Création du schéma sans relation OK

#### IV.V. Vérifions ce qui s'est passé en base de données

Cliquez sur le lien : [http://localhost/phpmyadmin/db\\_structure.php?server=1&db=cours](http://localhost/phpmyadmin/db_structure.php?server=1&db=cours)

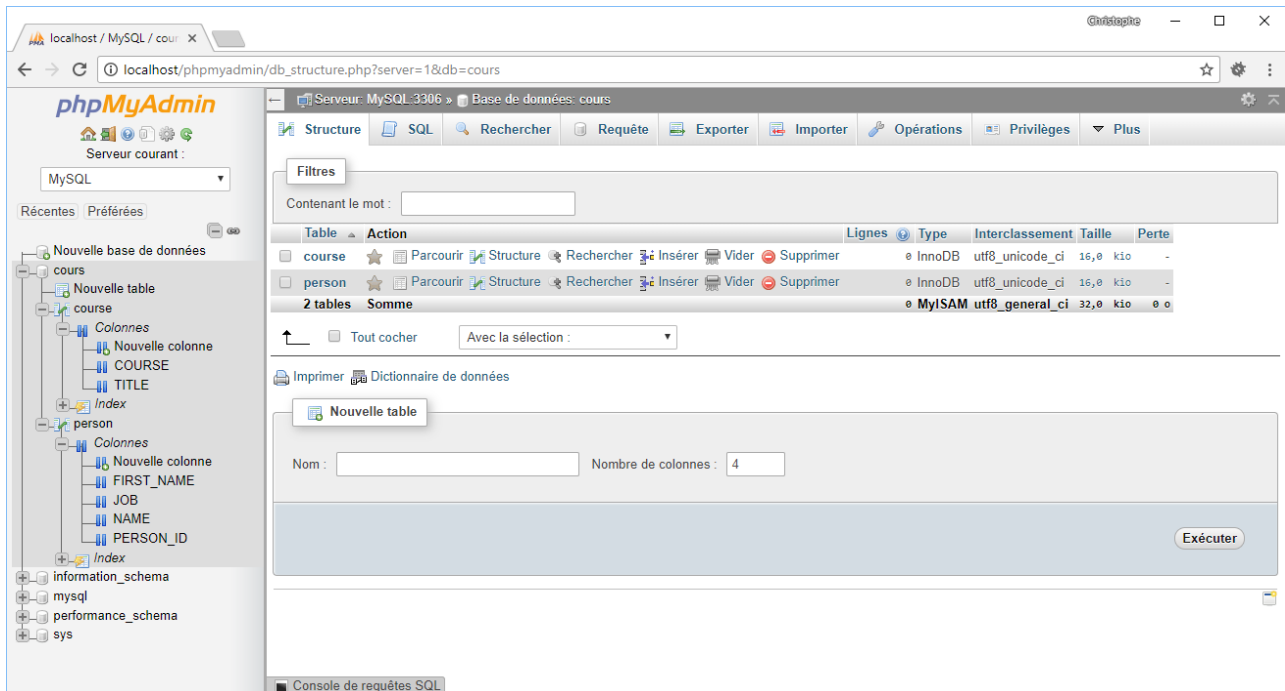


Figure 31 - Vérification de la base cours sans relation

## IV.VI. Mise en place des relations

### IV.VI.1 Relation ManyToOne

La première relation à positionner est celle du ManyToOne

```
<?php
namespace Entity;

/**
 * @Entity
 * @Table(name="COURSE")
 * @author Christophe PEQUIGNAT
 *
 */
class Course
{
    /**
     * @Id
```

```
* @Column(type="integer", name="COURSE")
* @GeneratedValue
* @var int
*/
private $id;

/**
 * @Column(type="string", length=1024, name="TITLE")
 * @var string
 */
private $title;

/**
 * @ManyToOne(targetEntity="Person")
 * @JoinColumn(name="AUTHOR_FK", referencedColumnName="PERSON_ID")
 */
private $author;
}
```

Nous allons réinitialiser la base de données :

```
vendor\bin\doctrine.bat orm:schema-tool:drop --force
```

```
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:drop --force
Dropping database schema...
[OK] Database schema dropped successfully!
c:\wamp64\www\Cours>
```

Figure 32 - Effacement de la base de données

Et nous relançons la création du schéma :

```

c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create

! [CAUTION] This operation should not be executed in a production environment!

Creating database schema...

[OK] Database schema created successfully!

c:\wamp64\www\Cours>

```

Figure 33 - Création de la base de données avec relation ManyToOne

Voyons dans l'outil HeidiSQL ce qui s'est passé.

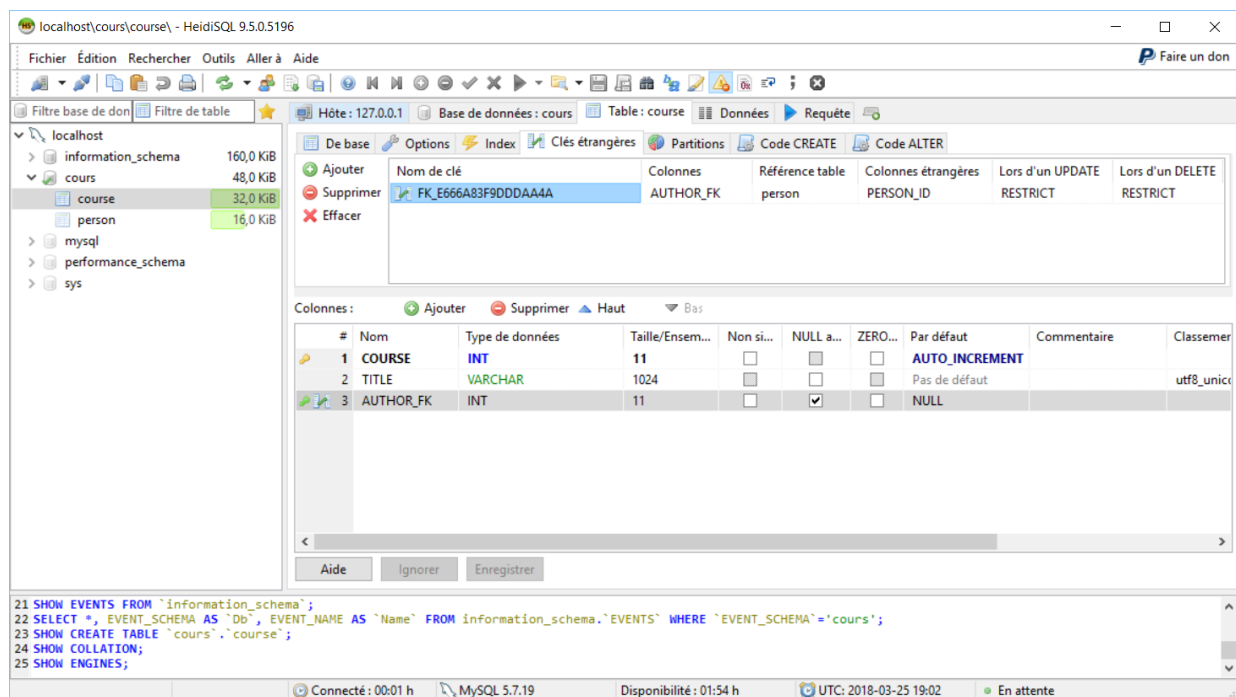


Figure 34 - Heidi SQL - Visualisation de la FK

## IV.VI.2 Relation inverse OneToMany

Nous allons rajouter dans l'Entity Person, la liste des cours dont il est author.  
Rajoutons dans la class Person

```

<?php

namespace Entity;

/**
 * @Entity
 * @Table(name="PERSON")

```

```
* @author Christophe PEQUIGNAT
*/
class Person
{
    /**
     * @Id
     * @Column(type="integer", name="PERSON_ID")
     * @GeneratedValue
     * @var int
     */
    private $id;

    /**
     * @Column(type="string", length=255, name="NAME")
     * @var string
     */
    private $name;

    /** @Column(type="string", length=255, name="FIRST_NAME")
     * @var string
     */
    private $firstname;

    /** @Column(type="string", length=255, name="JOB")
     * @var string
     */
    private $job;

    /**
     * Une Person rédige plusieurs Course.
     * @OneToMany(targetEntity="Course", mappedBy="author")
     */
    private $myCourses;
}
```

Nous modifions aussi le cours pour indiquer la double relation :

```
<?php
namespace Entity;

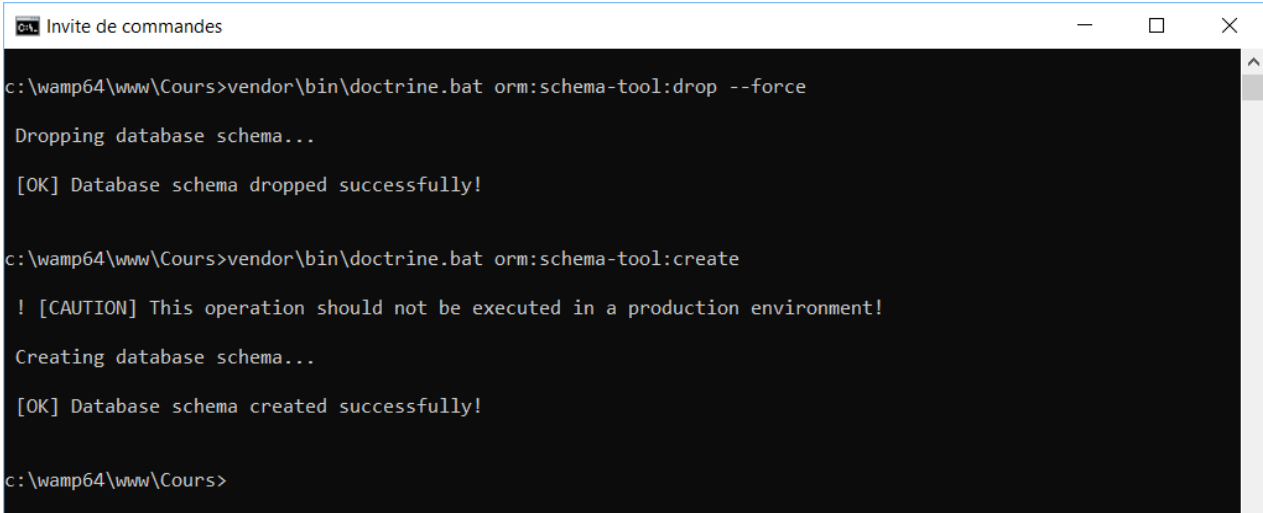
/**
 * @Entity
 * @Table(name="COURSE")
 * @author Christophe PEQUIGNAT
 *
 */
class Course
{

    /**
     * @Id
     * @Column(type="integer", name="COURSE")
     * @GeneratedValue
     * @var int
     */
    private $id;

    /**
     * @Column(type="string", length=1024, name="TITLE")
     * @var string
     */
    private $title;

    /**
     * @ManyToOne(targetEntity="Person", inversedBy="myCourses")
     * @JoinColumn(name="AUTHOR_FK", referencedColumnName="PERSON_ID")
     */
    private $author;
}
```

Rafraichissons la base de données.



```
Invite de commandes
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:drop --force

Dropping database schema...

[OK] Database schema dropped successfully!

c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create

! [CAUTION] This operation should not be executed in a production environment!

Creating database schema...

[OK] Database schema created successfully!

c:\wamp64\www\Cours>
```

Figure 35 - Recréation de la base de données

### IV.VI.3 Relation ManyToMany

Nous avons des Cours que suivent des Personnes. Cette relation est une relation ManyToMany. En effet, chaque cours peut être suivi par plusieurs personnes, et Une Personne peut suivre plusieurs Cours.

Modifions la Class Person.php

```
<?php
namespace Entity;

/**
 * @Entity
 * @Table(name="PERSON")
 * @author Christophe PEQUIGNAT
 */
class Person
{
    /**
     * @Id
     * @Column(type="integer", name="PERSON_ID")
     * @GeneratedValue
     * @var int
     */
}
```



```
    */
    private $id;

    /**
     * @Column(type="string", length=255, name="NAME")
     * @var string
     */
    private $name;

    /** @Column(type="string", length=255, name="FIRST_NAME")
     * @var string
     */
    private $firstname;

    /** @Column(type="string", length=255, name="JOB", nullable=true)
     * @var string
     */
    private $job;

    /**
     * Une Person rédige plusieurs Course.
     * @OneToMany(targetEntity="Entity\Course", mappedBy="author")
     */
    private $myCourses;

    /**
     * Plusieurs Persons suivent plusieurs cours.
     * @ManyToMany(targetEntity="Entity\Course", inversedBy="students")
     * @JoinTable(name="PERSON_COURSE",
     *             joinColumns={@JoinColumn(name="PERSON_ID",
     referencedColumnName="PERSON_ID")},
     *             inverseJoinColumns={@JoinColumn(name="COURSE_ID",
     referencedColumnName="COURSE_ID")}
     *             )
     */
    private $myTrainingCourses;
```

```
public function __construct() {  
    $this->myTrainingCourses = new  
    \Doctrine\Common\Collections\ArrayCollection();  
}  
  
}
```

Et la classe Course.php

```
<?php  
  
namespace Entity;  
  
/**  
 * @Entity  
 * @Table(name="COURSE")  
 * @author Christophe PEQUIGNAT  
 *  
 */  
  
class Course  
{  
  
    /**  
     * @Id  
     * @Column(type="integer", name="COURSE_ID")  
     * @GeneratedValue  
     * @var int  
     */  
    private $id;  
  
    /**  
     * @Column(type="string", length=1024, name="TITLE")  
     * @var string  
     */  
    private $title;
```

```
/**
 * @ManyToOne(targetEntity="Entity\Person", inversedBy="myCourses")
 * @JoinColumn(name="AUTHOR_FK", referencedColumnName="PERSON_ID",
 * nullable=false)
 */
private $author;

/**
 * Plusieurs cours ont plusieurs étudiants.
 * @ManyToMany(targetEntity="Entity\Person", mappedBy="myTrainingCourses")
 */
private $students;

public function __construct() {
    $this->students = new \Doctrine\Common\Collections\ArrayCollection();
}
}
```

On relance avec l'option `--dump-sql` pour la partie création afin d'avoir la requête générée :

```
c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:drop --force
Dropping database schema...
[OK] Database schema dropped successfully!

c:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create --dump-sql 1>..\install.sql

c:\wamp64\www\Cours>
```

Figure 36 - Installation ManyToMany DB

On récupère le fichier se trouvant dans `c:\wamp64\www\install.sql`

The following SQL statements will be executed:

```
CREATE TABLE COURSE (COURSE_ID INT AUTO_INCREMENT NOT NULL, TITLE
VARCHAR(1024) NOT NULL, AUTHOR_FK INT NOT NULL, INDEX IDX_E666A83F9DDDA4A
```

```
(AUTHOR_FK), PRIMARY KEY(COURSE_ID)) DEFAULT CHARACTER SET utf8 COLLATE
utf8_unicode_ci ENGINE = InnoDB;

CREATE TABLE PERSON (PERSON_ID INT AUTO_INCREMENT NOT NULL, NAME
VARCHAR(255) NOT NULL, FIRST_NAME VARCHAR(255) NOT NULL, JOB VARCHAR(255)
DEFAULT NULL, PRIMARY KEY(PERSON_ID)) DEFAULT CHARACTER SET utf8 COLLATE
utf8_unicode_ci ENGINE = InnoDB;

CREATE TABLE PERSON_COURSE (PERSON_ID INT NOT NULL, COURSE_ID INT NOT NULL,
INDEX IDX_63CA64765DF4E348 (PERSON_ID), INDEX IDX_63CA64762593919D (COURSE_ID),
PRIMARY KEY(PERSON_ID, COURSE_ID)) DEFAULT CHARACTER SET utf8 COLLATE
utf8_unicode_ci ENGINE = InnoDB;

ALTER TABLE COURSE ADD CONSTRAINT FK_E666A83F9DDDA4A FOREIGN KEY
(AUTHOR_FK) REFERENCES PERSON (PERSON_ID);

ALTER TABLE PERSON_COURSE ADD CONSTRAINT FK_63CA64765DF4E348 FOREIGN KEY
(PERSON_ID) REFERENCES PERSON (PERSON_ID);

ALTER TABLE PERSON_COURSE ADD CONSTRAINT FK_63CA64762593919D FOREIGN KEY
(COURSE_ID) REFERENCES COURSE (COURSE_ID);
```

## V. Mise en œuvre de l'accès à la base de données

### V.I. *Objectif*

L'objectif de cette partie est d'enrichir le site en mettant en place deux DAO (ou Repository) permettant l'accès respectivement au Cours et Personne.

### V.II. *Ajout des « Repository »*

Dans le répertoire Cours/src, rajouter un nouveau répertoire « Repository ».

Nous allons créer deux nouvelles classes : CourseRepository ainsi que PersonRepository

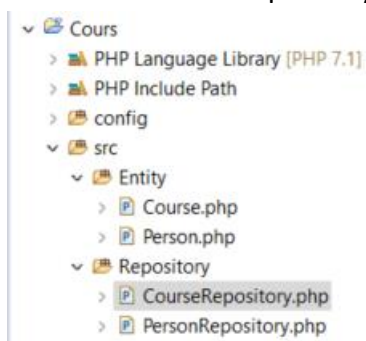


Figure 37 - Ajout des deux Repository

Les deux classes doivent étendre de « EntityRepository » de Doctrine\ORM\EntityRepository.

CourseRepository :

```
<?php
namespace Repository;

use Doctrine\ORM\EntityRepository;

/**
 *
 * @author Christophe PEQUIGNAT
 *
 */
class CourseRepository extends EntityRepository
{

    public function getAllCourses()
    {
        return $this->getEntityManager()->createQuery('SELECT c FROM
\Entity\Course c')
        ->getResult();
    }

    public function addCourse(\Entity\Course $course) {
        $this->getEntityManager()->persist($course);
    }

    public function updateCourse(\Entity\Course $course) {
        $this->getEntityManager()->persist($course);
    }

    public function removeAll() : void{
        foreach($this->getAllCourses() as $course) {
            $this->getEntityManager()->remove($course);
        }
    }
}
```

PersonRepository :

```
<?php
namespace Repository;

use Doctrine\ORM\EntityRepository;

class PersonRepository extends EntityRepository
{

    public function getAllPersons()
    {
        return $this->getEntityManager()->createQuery('SELECT p FROM
\Entity\Person p')
        ->getResult();
    }

    public function addPerson(\Entity\Person $person) : void{
        $this->getEntityManager()->persist($person);
    }

    public function removeAll() : void{
        foreach($this->getAllPersons() as $person){
            $this->getEntityManager()->remove($person);
        }
    }
}
```

Vous remarquerez que les requêtes d'accès à la base ne sont pas directement écrit en SQL.

```
SELECT p FROM \Entity\Person p
```

C'est volontaire afin de mettre une abstraction sur l'écriture pour permettre plus facilement la migration entre base de données. Aussi il est plus facile de changer de base entre MySQL, Posgres, Oracle, sqlite...

Les Repository à mon sens ne doivent pas gérer la transaction de la base de données. C'est au niveau au dessus que doivent être gérées ces transactions : dans la couche Service.

### V.III. *Mise à jour des Entity pour rajouter la déclaration des Repository*

Afin de pouvoir utiliser ces nouveaux Repository, ils doivent être reliés avec leur Entity.

Voici en jaune ce qui faut rajouter pour la prise en compte pour l'Entity Course :

```
<?php
namespace Entity;

use Doctrine\ORM\Mapping\Entity as Entity;

/**
 * @Entity(repositoryClass="Repository\CourseRepository")
 * @Table(name="COURSE")
 *
 * @author Christophe PEQUIGNAT
 */
class Course
{
    //...
}
```

Et pour l'Entity Person :

```
<?php
namespace Entity;

use Doctrine\ORM\Mapping\Entity as Entity;

/**
 * @Entity(repositoryClass="Repository\PersonRepository")
 * @Table(name="PERSON")
 *
 * @author Christophe PEQUIGNAT
```

```
*/  
class Person  
{  
//...  
}
```

Comme vous pouvez le voir, on utilise maintenant l'annotation `@Entity` de `Doctrine\ORM\Mapping\Entity`. Cela permet de conserver le mapping à la base ainsi que d'avoir les options avec le Repository : `repositoryClass="namespaces\\ClassRepository"`.

Il convient de rajouter maintenant les getters et setters dans les Entity.

#### V.IV. *Mise en place de la couche Service*

Maintenant, il convient de rajouter les classes permettant de répondre aux exigences métiers de l'application et ceci dans un contexte transactionnel. C'est la couche Service !

Créer un répertoire Service dans src.

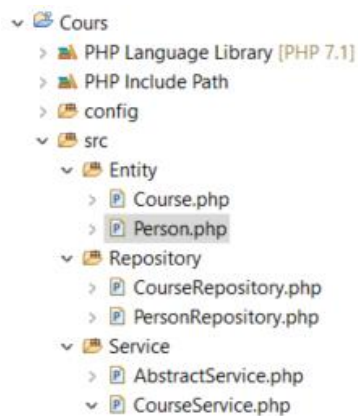


Figure 38 - Couche Service

Rajouter une classe abstraite : [AbstractService.php](#)

Cette classe sert à définir les méthodes et attribut commun à tous les Services de l'application : à savoir contenir l'EntityManager.

```
<?php  
namespace Service;  
  
use Doctrine\ORM\EntityManager;
```



```
abstract class AbstractService
{

    /**
     * @var EntityManager
     */
    protected $_em;

    protected function setEntityManager(EntityManager $em) : void
    {
        $this->_em = $em;
    }

    protected function getEntityManager() : EntityManager{
        return $this->_em;
    }

}
```

### Et la classe CourseService.php

```
<?php
namespace Service;

use Doctrine\ORM\EntityManager;
use Entity\Person;
use Entity\Course;

/**
 * La classe de service qui gère de haut niveau l'aspect métier ainsi que les
 * transactions
 *
 * Traitez les transactions au plus haut niveau possible pour mieux gérer les
 * plantages, oui cela arrive
 *
 * @author Christophe PEQUIGNAT
```

```
*
*/
class CourseService extends AbstractService
{

    public function __construct(EntityManager $em){
        $this->setEntityManager($em);
    }

    /**
     * Initialise la base de données avec des valeurs ceci dans une transaction
    unique
     */
    public function initExample() : void{
        $this->getEntityManager()->transactional(function(EntityManager $em){
            $author = new Person();
            $author->setFirstname("Christophe");
            $author->setName("Péquignat");
            $author->setJob("Software Architect");

            $course = new Course();
            $course->setAuthor($author);
            $course->setTitle('Mise en oeuvre de Doctrine 2.6');

            $author->addMyCourse($course);

            $student = new Person();
            $student->setFirstname("John");
            $student->setName("Doe");
            $student->setJob("Freelance");

            $em->getRepository('Entity\Person')->addPerson($author);
            $em->getRepository('Entity\Course')->addCourse($course);
            $em->getRepository('Entity\Person')->addPerson($student);
```

```
        $course->addStudent($student);
        $student->addMyTrainingCourse($course);
        //En sortie de la méthode le relation ManyToMany est enregistrée en
base
    });
}

/**
 * Efface la base de données de toutes ses valeurs ceci dans une transaction
unique
 */
public function dropExample() : void{
    $this->getEntityManager()->transactional(function(EntityManager $em){
        $em->getRepository('Entity\Course')->removeAll();
        $em->getRepository('Entity\Person')->removeAll();
    });
}

public function getAllPersons(){
    return $this->getEntityManager()->getRepository('Entity\Person')->
>getAllPersons();
}

public function getAllCourses(){
    return $this->getEntityManager()->getRepository('Entity\Course')->
>getAllCourses();
}
}
```

Cet exemple, ce n'est pas conforme pour une utilisation réelle, en effet nous créons les données de Tests directement dans le service. Hors il conviendrait d'écrire des Tests Unitaires ou utiliser un mécanisme en dehors de l'application web pour initialiser les données.

## V.V. Affichage des données

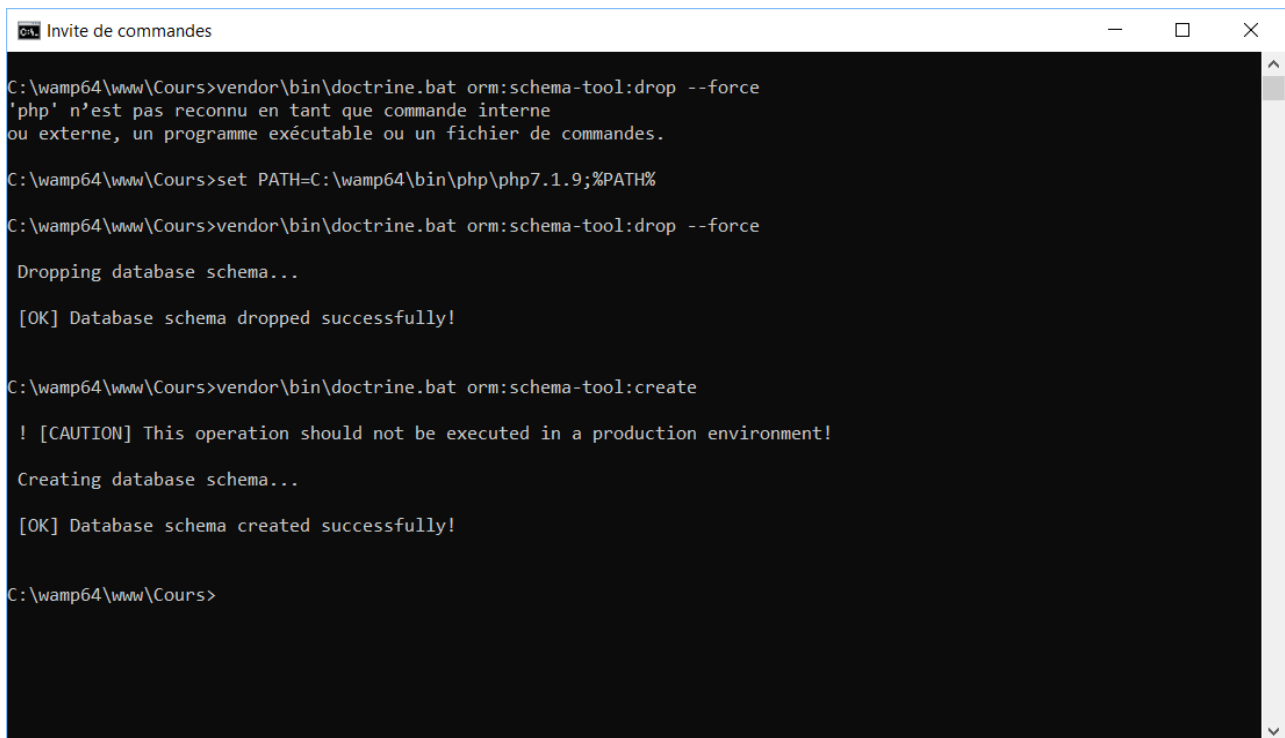
Maintenant que nous avons notre service d'appel, nous pouvons enfin réaliser l'affichage des données.

Voici le rendu lors que l'on lance la création de la base et ensuite une première fois la page <http://cours>

```
set PATH=C:\wamp64\bin\php\php7.1.9;%PATH%
```

```
vendor\bin\doctrine.bat orm:schema-tool:drop --force
```

```
vendor\bin\doctrine.bat orm:schema-tool:create
```



```
Invite de commandes

C:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:drop --force
'php' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\wamp64\www\Cours>set PATH=C:\wamp64\bin\php\php7.1.9;%PATH%

C:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:drop --force

Dropping database schema...

[OK] Database schema dropped successfully!

C:\wamp64\www\Cours>vendor\bin\doctrine.bat orm:schema-tool:create

! [CAUTION] This operation should not be executed in a production environment!

Creating database schema...

[OK] Database schema created successfully!

C:\wamp64\www\Cours>
```

Figure 39 - Réinitialisation de la base de données

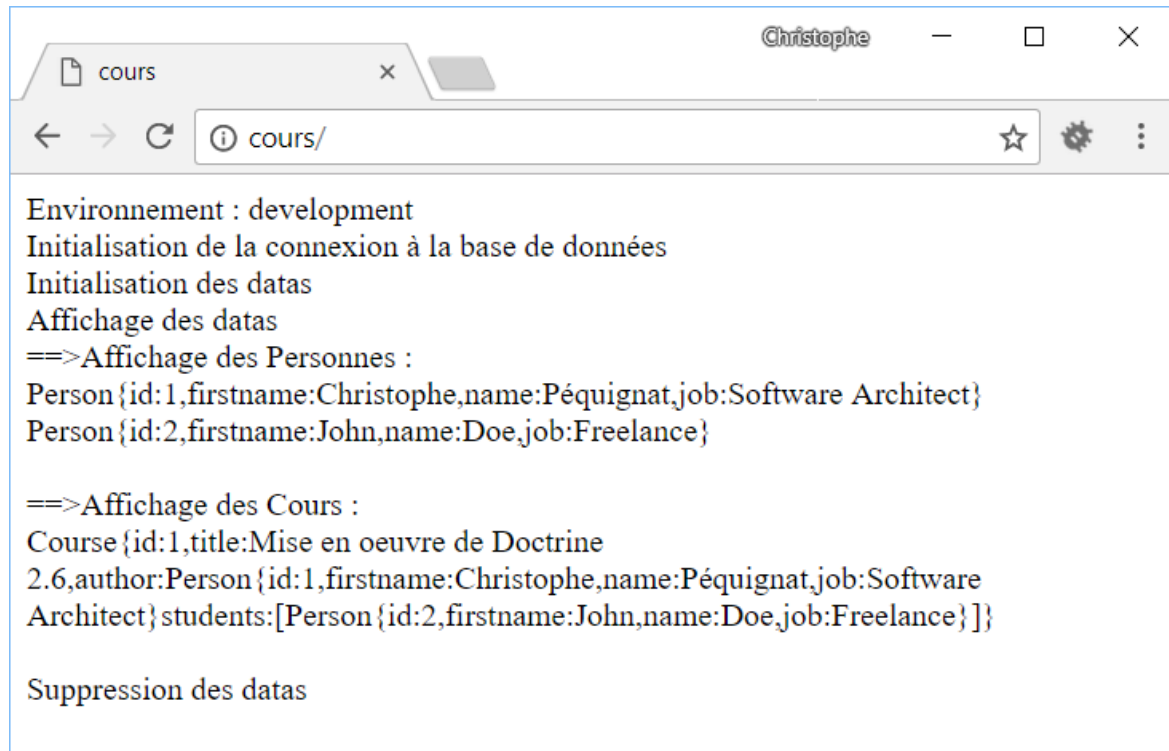


Figure 40 - Rendu simple mais efficace

### Le fichier `web/index.php`

```
<?php

defined ("APPLICATION_ENV") || define("APPLICATION_ENV",
getenv('APPLICATION_ENV') ?? 'production');

echo 'Environnement : ' . APPLICATION_ENV . "<br/>\r\n";

echo 'Initialisation de la connexion à la base de données'. "<br/>\r\n";
require_once('../bootstrap.php');

use \Service\CourseService;

$courseService = new CourseService(Bootstrap::getEntityManager());

echo 'Initialisation des datas'. "<br/>\r\n";
$courseService->initExample();
echo 'Affichage des datas'. "<br/>\r\n";

echo '==>Affichage des Personnes :'. "<br/>\r\n";
foreach ($courseService->getAllPersons() as $person) {
```

```
    echo $person."<br/>\r\n";  
}  
echo "<br/>\r\n";  
echo '==>Affichage des Cours :'. "<br/>\r\n";  
foreach($courseService->getAllCourses() as $cours){  
    echo $cours."<br/>\r\n";  
}  
echo "<br/>\r\n";  
echo 'Suppression des datas'. "<br/>\r\n";  
$courseService->dropExample();
```

## VI. Sources d'Informations

#	Source	Lien
[S1]	Doctrine	<a href="http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/">http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/</a>
[S2]	Doctrine Association - Mapping	<a href="http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html">http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html</a>

## VII. Fin du document