Doctrine

< Javascript >>

Doctrine est également un ORM qui peut être associé à CodeIgniter, il est beaucoup plus puissant, et plus complet.

- Site de référence Doctrine
- Documentation
- Téléchargement DoctrineORM-2.2.1-full

Installation

Doctrine est installé en tant que bibliothèque dans codeIgniter.

- Dézipper l'archive.
- Copier le dossier **Doctrine** de l'archive dans le dossier **application/libraries**.

Créer une classe Doctrine.php dans le dossier libraries :

|h application/libraries/Doctrine.php

```
<?php
class Doctrine
{
    // the Doctrine entity manager
    public $em = null;

    public function __construct()
    {
        // include our CodeIgniter application's database configuration
        require APPPATH.'config/database.php';
        // include Doctrine's fancy ClassLoader class
        require_once APPPATH.'libraries/Doctrine/Common/ClassLoader.php';

        // load the Doctrine classes
        $doctrineClassLoader = new \Doctrine\Common\ClassLoader('Doctrine',
APPPATH.'libraries');
        $doctrineClassLoader->register();

        // load Symfony2 helpers
        // Don't be alarmed, this is necessary for YAML mapping files
        $symfonyClassLoader = new \Doctrine\Common\ClassLoader('Symfony',
APPPATH.'libraries/Doctrine');
        $symfonyClassLoader->register();

        // load the entities
        $entityClassLoader = new \Doctrine\Common\ClassLoader('Entities',
APPPATH.'models');
        $entityClassLoader->register();
```

```
        // load the proxy entities
        $proxyClassLoader = new \Doctrine\Common\ClassLoader('Proxies',
APPPATH.'models');
        $proxyClassLoader->register();

        // set up the configuration
        $config = new \Doctrine\ORM\Configuration;

        if(ENVIRONMENT == 'development')
            // set up simple array caching for development mode
            $cache = new \Doctrine\Common\Cache\ArrayCache;
        else
            // set up caching with APC for production mode
            $cache = new \Doctrine\Common\Cache\ApcCache;
        $config->setMetadataCacheImpl($cache);
        $config->setQueryCacheImpl($cache);

        // set up proxy configuration
        $config->setProxyDir(APPPATH.'models/Proxies');
        $config->setProxyNamespace('Proxies');

        // auto-generate proxy classes if we are in development mode
        $config->setAutoGenerateProxyClasses(ENVIRONMENT == 'development');

        // set up annotation driver
        //$yamlDriver = new
\Doctrine\ORM\Mapping\Driver\YamlDriver(APPPATH.'models/Mappings');
        $driverImpl = $config->newDefaultAnnotationDriver(APPPATH.'models');
        $config->setMetadataDriverImpl($driverImpl);

        // Database connection information
        $connectionOptions = array(
            'driver' => 'pdo_mysql',
            'user' => $db['default']['username'],
            'password' => $db['default']['password'],
            'host' => $db['default']['hostname'],
            'dbname' => $db['default']['database']
        );

        // create the EntityManager
        $em = \Doctrine\ORM\EntityManager::create($connectionOptions,
$config);

        // store it as a member, for use in our CodeIgniter controllers.
        $this->em = $em;
    }
}
?>
```

Doctrine doit être ensuite chargé automatiquement avec autoload.php :

```
$autoload['libraries'] = array('database','doctrine');
```

Le chargement de la bibliothèque **database** est indispensable

Logiquement, Doctrine est prêt à fonctionner.
Vérifier que la page d'accueil ne produit pas d'erreurs : http://localhost/doctrine_CI/

Création des classes métier

Une classe métier correspond à la notion d'**entity** dans Doctrine.

Considérons la base de données suivante :

La base de données sera composée de 2 entities: utilisateur et categorie. La relation de type CIF entre utilisateurs et categories peut s'exprimer de la façon suivante :

- Chaque utilisateur appartient à 1 catégorie (belongs_to)
- Dans chaque categorie, on peut compter de 0 à n utilisateurs (has_many)