Document ORM Mélanie2 PHP

Ce document décrit le fonctionnement et la mise en place de l'ORM Mélanie2 en PHP.

Sommaire

Versions

V0.1	Présentation de la librairie ORM Mélanie2
V0.2	Ajout de nouvelles informations sur le fonctionnement de l'ORM
V0.3	Corrections
V0.4	Modification pour diffusion publique de l'ORM

Présentation

Définition ORM

Un ORM (object-relationnal mapping) va permettre de faire le lien entre une base de données et des objets. Ces objets seront formatés de façon à être facilement exploitable par les applications.

Notre librairie est un ORM écrit en PHP qui permet le mapping de la base de données Horde Mélanie2.

Intérêt de cette librairie pour Mélanie2

La base de données Mélanie2 (Horde) a un schéma très spécifique qui n'a pas évolué depuis des années. Or, de plus en plus d'applications de présentation et de synchronisation utilisent cette base de données. L'idée est donc de faciliter le développement de ces applications en proposant des méthodes de développement simple pour l'accès à ces données. De plus des API de type service Web peuvent être proposé afin d'implémenter cette librairie pour d'autres langages que le PHP.

Installation

Récupération de la librairie

La librairie ORM Mélanie2 peut être récupérée auprès du PNE Annuaire et Messagerie du MEDDE ou bien depuis les sources git (disponibles sur github ou sur le serveur git du PNE).

La version peut ensuite être décompressée dans un répertoire.

Configuration du fichier php.ini

Ajouter dans la configuration du php.ini (cli ou apache2) le chemin vers la librairie dans le champs "include_path".

Exemple:

> ; UNIX: "/path1:/path2"

> include_path = ".:/usr/share/php:/usr/share/libM2/LibrarySqlMelanie2"

Bien inclure le répertoire "LibrarySqlMelanie2/".

Documentation PHPDocs

La documentation se trouve dans le répertoire "docs_html/". Il s'agit d'une documentation générée via phpdocs. On peut la consulter en ouvrant le fichier "index.html" dans un navigateur.

Mise en production

Depuis la version 0.1.7 de l'ORM une configuration externe est possible. Le chemin vers la configuration externe est éditable dans le fichier config/env.php (par défaut /etc/LibM2) :

```
/**
 * Chemin vers la configuration externe
 */
define('CONFIGURATION_PATH_LIBM2', '/etc/LibM2');
```

De plus la configuration peut permettre de gérer plusieurs applications indépendantes ayant une configuration différentes. Dans ce cas il faut définir un type « external » dans le fichier config/env.php :

```
/**
 * Configuration externe ou interne
 * La configuration TYPE_INTERNAL va lire les données dans le répertoire
/config de l'ORM
 * Dans ce cas la configuration chargée sera fonction du ENVIRONNEMENT_LIBM2
 * La configuration TYPE_EXTERNAL va les lire les données dans un répertoire
configuré dans CONFIGURATION_PATH_LIBM2
 */
define('CONFIGURATION_TYPE_LIBM2', TYPE_EXTERNAL);
```

Pour la production, les fichiers de configuration "config/ldap.php" et "config/sql.php" doivent être paramétrés.

Les répertoires "docs_html/", "documentation/", "tests/" et le fichier "example.php" peuvent être supprimés.

Licence

L'ORM Mélanie2 est distribuée sous licence GPLv3 (http://www.gnu.org/licenses/gpl.html)

Points sensibles

Serveur

La librairie ORM M2 ne nécessite pas de module particulier. Le seul besoin étant de disposer d'un serveur Apache avec PHP >= 5.3 installé. Le PHP 5.3 apporte notamment la gestion des namespaces ainsi que l'implémentation améliorée des méthodes magiques.

Il semblerait qu'une version >= 5.4 de PHP améliore les performances de ces méthodes magiques (qui sont obligatoirement moins performantes que les méthodes classiques).

Version actuelle

La version actuelle en développement est la 0.1.8. La version en production sur les produits Mélanie2 est la 0.1.7.

Les produits utilisant l'ORM Mélanie2 en production actuellement sont Zpush2 et Roundcube. Un développement de serveur CalDAV via SabreDAV est également en cours.

Application

Le but de cette librairie étant de simplifier l'implémentation du backend Mélanie2 dans les applications, il devrait être possible de l'intégrer dans a peu près n'importe quelle application (écrite en PHP ou via des API REST/WS). Néanmoins quelques pré-requis devraient permettre de faciliter l'implémentation :

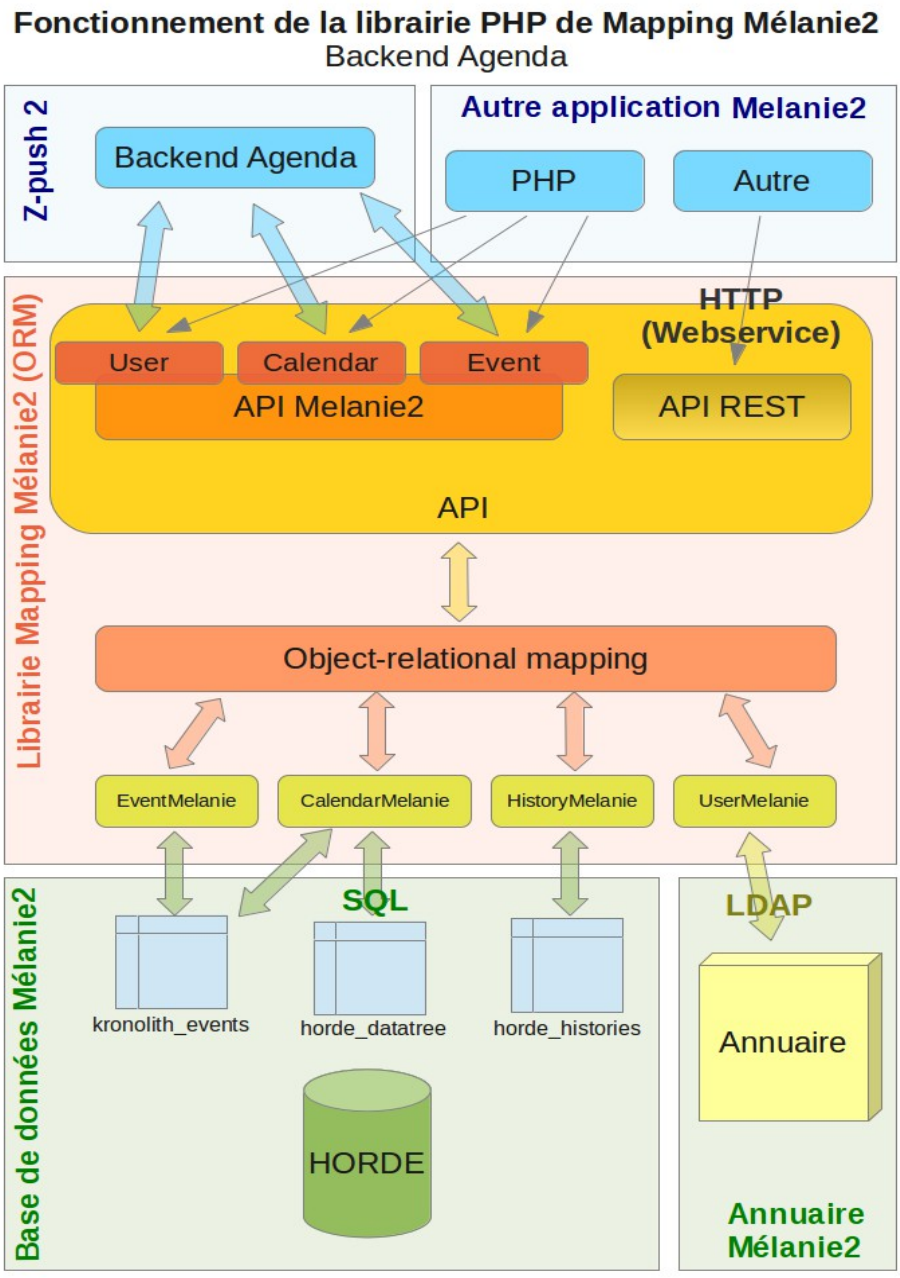
- La séparation entre le backend de données et la présentation au sein de l'application. Normalement les applications récentes permettent de choisir différents backends pour le stockage des données. Ils sont donc a priori séparés du reste de l'application ce qui permet de simplifier les modifications. Dans le cas contraire, il sera nécessaire

d'identifier tous les appels au backend pour les remplacer par des appels à la librairie.

- La gestion des évènements doit se rapprocher au maximum de celle dans horde/kronolith. Actuellement la base de données Mélanie2 ne stocke pas beaucoup d'informations sur la gestion des évènements, notamment au niveau de la récurrence. Un haut niveau de configuration des évènements dans la nouvelle application pourrait être problèmatique sans évolution de la base de données Mélanie2.

Présentation technique

Schéma fonctionnel partie Backend Agenda avec Zpush2



Ce schéma représente l'architecture de l'ORM pour la partie Agenda. Tous les modules sont basés sur le même principe, seul le noms des champs dans la base de données et les noms d'objets sont différents.

Couche d'extraction des données

Les données sont extraites en SQL depuis la base de données ou en LDAP depuis l'annuaire Mélanie2. Les besoins couverts sont donc assez large, allant de l'authentification de l'utilisateur à la liste des évènements dans un laps de temps.

La couche d'abstraction des données utilise la librairie native PDO (http://php.net/manual/fr/book.pdo.php). Pour permettre le mapping des données, les données sont extraites via PDO en mode « fetch to classe » qui aura pour effet d'instancier une nouvelle classe avec les données reçu de la base.

Pour s'éviter d'implémenter une classe par table en définissant exactement le schéma dans celle-ci, on utilise une classe abstraite qui implémente les méthodes magiques de php (http://php.net/manual/fr/language.oop5.magic.php). Cette classe (`LibMelanie\Lib\MagicObject`) est abstraite et est étendue par tous les objets et conteneur de la librairie. Un premier mapping au niveau du nom des attributs est alors exécuté au moment de la lecture des données via les méthodes magiques __get et __set.

Couche de mapping des données

Le mapping des valeurs se fait dans un deuxième temps via une deuxième classe abstraite utilisant des méthodes magiques (`LibMelanie\Lib\Melanie2Object`). Ce sont alors les classes d'API Mélanie2 qui implémente cette classe. Cette séparation a été faite pour séparer la couche d'extraction des données de celle de présentation des données aux applications. L'idée sera alors de simplifier le changement vers une nouvelle base de données par exemple.

Configuration du mapping

Le mapping se configure en premier lieu dans le fichier config/mapping.php. Il permet d'associer les noms de propriétés des objets aux nom des champs dans la base de données. Il s'agit du mapping bas niveau effectué par la classe `LibMelanie\Lib\MagicObject`.

Par exemple :

```
// Gestion des évènements : objet EventMelanie
"EventMelanie" => array(
            "uid" => array(self::name => "event_uid", self::type =>
self::string, self::size => 255),

            // DATA
            "title" => array(self::name => "event_title", self::type =>
self::string, self::size => 255, self::defaut => ''),
```

Pour le mapping plus spécifique, nécessitant plus de code notamment pour la conversion des données, il est possible de définir dans les classes des méthodes « setMap<*Property*> »/« getMap<*Property*> » qui seront automatiquement appelées lors de la lecture ou la définition d'une propriété de la classe.

Par exemple :

```
/**
 * Mapping status field
 * @param Event::STATUS_* $status
 */
protected function setMapStatus($status) {
	M2Log::Log(M2Log::LEVEL_DEBUG, get_class($this)."-
>setMapStatus($status)");
	if (!isset($this->objectmelanie)) throw new
Exceptions\ObjectMelanieUndefinedException();
	if (isset(MappingMelanie::$MapStatusObjectMelanie[$status]))
		$this->objectmelanie->status = MappingMelanie::
$MapStatusObjectMelanie[$status];
}
/**
 * Mapping status field
 */
protected function getMapStatus() {
	M2Log::Log(M2Log::LEVEL_DEBUG, get_class($this)."->getMapStatus()");
	if (!isset($this->objectmelanie)) throw new
Exceptions\ObjectMelanieUndefinedException();
	if (isset(MappingMelanie::$MapStatusObjectMelanie[$this->objectmelanie-
>status]))
		return MappingMelanie::$MapStatusObjectMelanie[$this-
>objectmelanie->status];
	else
		return MappingMelanie::
$MapStatusObjectMelanie[self::STATUS_CONFIRMED];
}
```

Exemple d'implémentation

Voici un exemple d'implémentation de la librairie ORM M2. Cet exemple est basé sur le fichier example.php disponible à la racine du code.

Inclusion des fichiers

```
// Inclure le fichier includes.php
include_once 'includes/includes.php';
```

Cette inclusion se base sur le fait que la librairie se trouve bien dans le include_path du php.ini

La librairie utilise des namespaces, pour pouvoir utiliser les classes il est donc utile d'appeler les bons namespace, quelques exemples :

```
// API
use LibMelanie\Api\Melanie2\User;
use LibMelanie\Api\Melanie2\Calendar;
use LibMelanie\Api\Melanie2\Event;
```

La configuration des logs permet d'ajouter les logs de la librairie directement dans les logs de l'application. Il est également possible d'écrire dans un fichier de log indépendant, tout se configure au niveau de l'application.

```
// Log
use LibMelanie\Log\M2Log;

// Configurer les LOG
$log = function ($message) {
      echo "[LibM2] $message \r\n";
};
M2Log::InitDebugLog($log);
M2Log::InitInfoLog($log);
M2Log::InitErrorLog($log);
```

Pour l'instant il existe 3 niveaux de logs : Info, Debug et Error, d'autres pourront peut être arriver ensuite.

Il est ensuite possible d'instancier des objets conteneurs, par exemple :

```
// Définition d'un utilisateur
$user = new User();
$user->uid = 'julien.test1';

// Définition des conteneurs de l'utilisateur
$calendar = new Calendar($user);

// ID des conteneurs
$calendar->id = 'julien.test1';

// Charger les conteneur
$calendar->load();
```

Chaque conteneur s'instancie exactement de la même façon.

Il est également possible de vérifier les droits pour chaque conteneur.

```
// Vérifier les droits
echo $calendar->asRight(ConfigMelanie::READ) ? "calendar->read OK" : "calendar->read NOK";
```

On peut également lister les éléments d'un conteneur

```
// Lister tous les objets d'un conteneur
$events = $calendar->getAllEvents();
```

Les objets suivent le même principe d'implémentation. Par exemple pour le chargement

```
// Récupération d'un objet
$event = new Event($user, $calendar);
$event->uid = 'c2926c54-17af-4753-a614-3589eca2644e';
$event->load();
```

Pour les modifications/créations d'objets

```
// Modifier un objet
$event->title = "Mon titre d'évènement";
$event->save();
```

Ensuite quelques spécificités liées aux évènements (participants, récurrence)

```
// Gestion d'un evenement
$attendees = $event->attendees;
if ($attendees[0]->response == ConfigMelanie::ACCEPTED) echo "Attendee accepted";
$event->recurrence->days = array(Recurrence::RECURDAYS_MONDAY, Recurrence::RECURDAYS_SUNDAY);
```

Toutes les propriétés et méthodes sont visibles depuis la documentation PHPDocs ou accessibles en auto-complétion via un IDE comme eclipse par exemple.