Inf340 Systèmes d’information

Quatrième site

Objectifs

Objectif :

- Démonstration de ce que vous aurez à faire en TP
- Présentation de Doctrine

- Le site permet de gérer des galeries d’images.

Bilan du troisième site

Nous avions un site :

- Avec une gestion de la sécurité
- Qui utilise des composants existants
- Facilement déployable
- Simple à maintenir

Mais ActiveRecord contraint la base : pas de clefs multiples + un id par table.

ORM

Un mapping objet-relationnel (en anglais object-relational mapping ou ORM) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. On pourrait le désigner par « correspondance entre monde objet et monde relationnel » (Wikipedia).

ORM

- A une classe (une entité) correspond une table
- A un attribut correspond une colonne

- Possibilité d’avoir des cycles de vie (Triggers)

Description des entités

Soit au format XML, soit au format YAML, soit avec des annotations, nous utiliserons ces dernières.

```
/**
 * Utilisateur
 *
 * @Table(name="utilisateur", uniqueConstraints={@UniqueConstraint(columns={"description"})})
 * @Entity (repositoryClass="\models\repositeries\UtilisateurRepository")
 */
class Utilisateur
{
/**
 * @var string $login
 *
 * @Column(name="login", type="string", length=45, nullable=false)
 * @Id
 * @GeneratedValue(strategy="NONE")
 */
private $login;

/**
 * @var integer $level
 *
 * @Column(name="level", type="integer", nullable=true)
 */
private $level;
…
```

Possibilité de règles de visibilité

Outre les contraintes de mapping objet relationnel, il est possible d'exprimer des contraintes de visibilité.

Par exemple un utilisateur peut voir l'ensemble de ses images.

```
/**
 * @OneToMany(targetEntity="Image",
   mappedBy="utilisateur")
 */
private $images;
```

Possibilité de cycle de vie

Lorsque l'on détruit un utilisateur, il faut détruire ces images et ce non seulement dans la base mais aussi sur le système de gestion de fichier.

```
/**
 * Image
 *
 * @Table(name="image")
 * @Entity(repositoryClass="\models\repositeries\ImageRepository")
 * @HasLifecycleCallbacks
 */
class Image
…
/**
 * @preRemove
 */
public function deleteFile()
{
…
```

Les dépôts

Les entités possèdent un repository par défaut qui contient les méthodes suivantes :

- public function findAll()
- public function findBy(array $criteria)
- public function findOneBy(array $criteria)
- …

Il est possible d'hériter de EntityRepository est de redéfinir ces méthodes.

Nous utiliserons les repository comme des composants métiers en ajoutant les méthodes d'accès à notre modèle.

Les dépôts

Dans l'entité : @Entity (repositoryClass="\models\repositeries\UtilisateurRepository")

Dans le dépôt : class UtilisateurRepository extends \Doctrine\ORM\EntityRepository {

```
public function create($login, $level, $password, $description) {
    …
    }

}
```

Les dépôts sont accessibles via le nom de l'entité :

- $em = **$this->doctrine->em;**
- $repository = $em->getRepository('models\Utilisateur');
- $repository->create($login, $password, $level, $description);

Le gestionnaire d'entité

- EntityManager est la classe qui a pour responsabilité de gérer la persistance des objets.
- Pour nous :
  - Dans un contrôleur qui a chargé doctrine : $em = $this->doctrine->em;
  - Dans un dépôt : $em = $this->getEntityManager();

Création

- Construire un objet puis le rendre persistant et réaliser l'écriture :

```
$utilisateur1 = new models\Utilisateur('admin', 0,
    'pass', 'le chef');
$utilisateur2 = new models\Utilisateur('toto', 1,
    'toto', 'pas le chef');
$em->persist($utilisateur1);
$em->persist($utilisateur2);
$em->flush();
```

Suppression

- Trouver l’entité puis la supprimer :

```
$utilisateur = $this->findOneByLogin($login);
$em->remove($utilisateur);
$em->flush();
```

Modification

- Trouver l'entité, la modifier puis la sauvegarder.

```
$utilisateur = $this->findOneByLogin($login);

$utilisateur->setPassword($password);
$utilisateur->setLevel($level);
$utilisateur->setDescription($description);
$em->persist($utilisateur);
$em->flush();
```

Interrogation

- En utilisant find :
  - Find
  - FindAll
  - FindOne
  - FindOneBy

Autre requêtes

- DQL : Un langage de requêtes objet
  - $max= $em->createQuery('SELECT max(i.url) FROM models\Image i')->getSingleScalarResult();
- Des requêtes natives
  - $em->getConnection()->exec('alter table image auto_increment=1');

Limitations rencontrées

- L’impossiblité d’avoir des associations n-n paramétrées.

- La gestion des champs autoincrementant avec mysql pas de séquence et pas de table doctrine pour les gérer.