Programmation Web sur le Serveur

Mini projet Tweeter (partie 1 - Les modèles)

Amine boumaza

amine.boumaza@univ-lorraine.fr

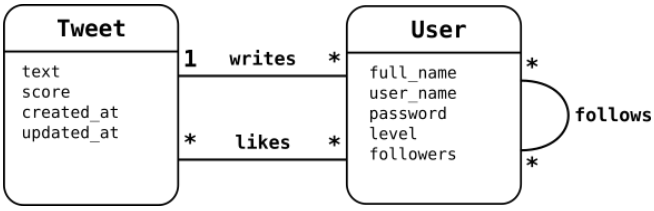
Modèles pour Tweeter

Le but de cette première phase est de s'occuper de la partie qui gère les données dans notre application. Il s'agit de coder les différentes classes qui communiquent avec le SGBD et qui permettent d'avoir une couche d'abstraction.

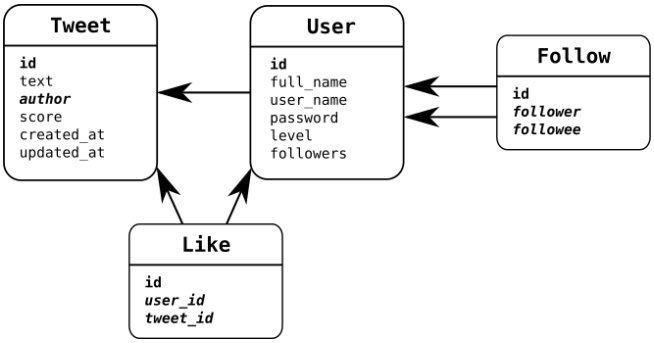
D'après les fonctionnalités de l'application nous définissons les données suivantes :

- les Tweets qui sont rédigés et évalués par des utilisateurs. Ils sont représentés par :
  - un texte, une date de publication une date de modification et un score.
- les Utilisateurs qui rédigent et évaluent les Tweets, et suivent d'autre utilisateurs. Ils sont représentés par :
  - un nom long, un nom d'utilisateur, un mot de passe, un niveaux d'accès et un nombre de suiveurs.

Ses données sont représentée par les entités suivantes :



Ce modèle des données ce traduit par les relations suivantes :



Dans ce qui suit, nous allons illustrer le fonctionnement et l'utilisation d'eloquent en créant les modèles de note application Tweeter.

Utilisation de l'ORM Eloquent

Pour les besoins de ce projet, nous allons utiliser l'ORM Eloquent, qui facilitera la conception des objets modèle. Dans ce qui suit, on présente une rapide introduction à cette librairie, la documentation officielle est beaucoup plus complète.

Installation

Afin d'utiliser l'ORM *Eloquent*, il est nécessaire de l'installer dans la racine de votre projet. L'installation se fait grâce à l'outil composer qui permet d'installer des applications ou des librairies depuis packagist, un dépôt d'outils pour les applications Web.

Pour *Eloquent*, il faut installer la librairie **`illuminate/database`** depuis la ligne de commande dans la racine du projet comme suit :

```
terminal> composer require illuminate/database
```

Une fois l'installation terminée, un répertoire nommé **vendor** est créé dans le répertoire courant. Ce répertoire contient *Eloquent* et tous les outils dont il a besoin.

1. Installer *Eloquent* à la racine du projet (noter l'apparition des fichiers **composer.json** et **composer.lock**

Connexion au SGBD

La procédure de connexion à la base de donnée est très similaire à celle de **PDO**. Les paramètres de connexion peuvent aussi être stockés dans un fichier de configuration lu par la fonction **parse_ini_file()**.

```php
<?php

/* pour le chargement automatique des classes d'Eloquent (dans le répertoire vendor) */
require_once 'vendor/autoload.php';

$config = [ /* ces informations doivent être dans un fichier ini */
    'driver'    => 'mysql',
    'host'      => 'localhost',
    'database'  => 'database',
    'username'  => 'user',
    'password'  => 'password',
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '' ];

/* une instance de connexion  */
$db = new Illuminate\Database\Capsule\Manager();

$db->addConnection( $config ); /* configuration avec nos paramètres */
$db->setAsGlobal();            /* rendre la connexion visible dans tout le projet */
$db->bootEloquent();           /* établir la connexion */
```

1. Mettre les paramètres de connexion dans le fichier **conf/config.ini** ;
2. Dans le script principal (**main.php**), charger les paramètres, configurer et établir une connexion.

Définition des modèles

Dans ce qui suit, nous supposerons que nous disposant d'une base de donnée avec deux tables *ville* et *club* avec les mêmes associations tel que nous les avons définit en cours.

Le principe de base de l'utilisation d'*Eloquent* est de programmer les objets modèle de l'application en sous-classant une classe générique (**\Illuminate\Database\Eloquent\Model**) qui fournit toutes les méthodes nécessaires pour interagir avec la base de données.

En héritant de la classe **\Illuminate\Database\Eloquent\Model**, la classe du modèle doit définir les attributs **$table** et **$primaryKey**, qui doivent renseigner le nom de la table et le nom de la clé primaire.

Si l'attribut **$timestamp** est vrai, la table doit contenir deux colonnes (nommées **updated_at**, **created_at**) qui sont mises-à-jour à chaque modification d'un enregistrement. Par exemple :

```php
<?php

class Ville extends \Illuminate\Database\Eloquent\Model {

       protected $table      = 'ville';  /* le nom de la table */
       protected $primaryKey = 'id';     /* le nom de la clé primaire */
       public    $timestamps = false;    /* si vrai la table doit contenir
                                            les deux colonnes updated_at,
                                            created_at */
}
```

1. Télécharger le fichier tweeter.sql afin de créer les relations sur le SGBD ;
2. Écrire les classes des modèles (**User**, **Tweet**, **Follow**, **Like**), elles appartient toutes à l'espace de nom **\tweeterapp\model**. (**attention** : seule la table **tweet** utilise les deux colonnes **updated_at** et **created_at**) ;
3. Tester que les classes modèles peuvent bien être instanciées depuis le script principal.

Exécuter des requêtes

Les requêtes sont construites en appelant successivement des méthodes de construction de requêtes. Ces méthodes retournent une requête qui sera ensuite exécutée pour récupérer le résultat. Le schéma classique pour faire une requête est donc le suivant :

1. Construire la requête en éventuellement plusieurs étapes (voir ci-dessous),
2. L'exécuter avec :
    - *la méthode **get** qui retourne le résultat sous la forme d'une collection d'objets du modèle (plusieurs lignes de la table)*
    - *ou la méthode **first** qui retourne un objet du modèle (une ligne de la table)*

Récupérer des données

Les requête simples sont construites avec la méthode **select** par exemple :

```php
<?php

$requete = Ville::select();  /* SQL : select * from 'ville' */
$lignes = $requete->get();   /* exécution de la requête et plusieurs lignes résultat */

foreach ($lignes as $v)      /* $v est une instance de la classe Ville */
       echo "Identifiant = $v->id, Nom = $v->nom\n" ;
```

1. Dans le script principal, exécuter les requêtes suivantes pour les tester:
    - Récupérer tous les utilisateurs ;
    - Récupérer la liste de tous les Tweets.

Récupérer des données avec des critères

La méthode **where** permet de spécifier des critères et peut être utilisées comme suit :

```php
<?php

$requete = Ville::select()->where('id', '=', 10);   /* SQL :  select * from 'ville' where 'id'=10 */
$v = $requete->first();                             /* exécution de la requête et une linge résultat *

echo "Identifiant = $v->id, Nom = $v->nom\n" ;      /* $v est une instance de la classe Ville*/
```

Les résultat peuvent être ordonnés avec la méthode **orderBy** comme suit :

```php
<?php

$lignes = Club::select('nom', 'sport')
                  ->where('nom', 'like', 'Nancy%')
                  ->orderBy('sport')
                  ->get();

foreach ($lignes as $c)
       echo "Discipline = $c->sport, Club = $c->nom\n" ;
```

1. Dans le script principal, exécuter les requêtes suivantes :
    - Récupérer la liste de tous les Tweets et les ordonner par date de modification. ;
    - Récupérer la liste des Tweets qui ont un score positif ;

Une liste exhaustive des méthodes pour la définition des critères et l' ordonnancement des résultat est disponible sur la documentation officielle.

Eloquent traduit les requêtes exprimées sous la forme d'appels successifs de méthodes en requêtes SQL en se basant sur les information des modèle (l'attribut **`$table`** et **`$primaryKey`**). La requête construite et une requête préparée en PDO qui est en suite exécutée avec la méthode **`get()`** ou **`first()`** .

```php
<?php

Club::where('id', '=', 12)->first()
```

```sql
/* SQL */
select * from 'club' where id = ? limit 1
/* avec ? remplacé par 12 */
```

Modifier les données

La modification (insertion, suppression et modification) des enregistrements en base de donnée est effectuée en modifiant les attributs d'une instance du modèle. La sauvegarde dans la base est effectuer avec la méthode **`save()`**.

```php
<?php

// insertion
$v = new Ville();
$v->nom = 'Nancy';
$v->save();

// mise-à-jour
$v->pays = 'France';
$v->save();

// suppression
$v->delete();
```

```sql
/* SQL */

insert into 'ville' ('nom') values (?)
/* avec ? remplacé par 'Nancy' */



update 'ville' set 'pays' = ? where 'id' = ?
/* avec ? remplacé par 'france' et 5 */
/* d'où vient le 5 ? */

delete from 'ville' where 'id' = ?
/* avec 5 */
```

1. Dans le script principal, exécuter les requêtes suivantes :
   - Ajouter un Tweet à la base de données (ne pas oublier de mettre un identifiant existant pour l'auteur) ;
   - Ajouter un nouvel utilisateur à la liste des utilisateurs.

La gestion des associations

Les associations entre les relations sont définies dans le modèle en codant les méthodes qui retourne le modèle associé.

Association 1->*

Pour les associations 1->* (une ville contient plusieurs clubs), on défini une méthode **`clubs()`** (notez le pluriel) dans la classe **`Ville`** qui retourne une collection de d'objets **`Club`**. Inversement, on défini une méthode **`ville()`** (notez le singulier) dans la classe **`Club`** qui retourne un objet **`Ville`**.

Eloquent fourni deux méthodes pour réaliser cela, qui sont respectivement **`hasMany()`** et **`belongsTo()`**.

Dans la classe **`Ville`** :

```php
<?php

/* dans le modèle Ville */

public function clubs() {
       return $this->hasMany('Club', 'ville_id');
       /* 'Club'     : le nom de la classe du modèle lié   */
       /* 'ville_id' : la clé étrangère dans la table liée */
}

/* Ailleurs ...pour  récupérer les clubs d'une ville donnée */

$v = Ville::where('id', '=', 2)->first();
$liste_clubs = $v->clubs()->get() ;
```

Dans la classe **`Club`** :

```php
<?php

/* dans le modèle Club */

public function ville() {
       return $this->belongsTo('Ville', 'ville_id');

       /* 'Ville'    : le nom de la classe du model lié */
       /* 'ville_id' : la clé étrangère dans la table courante */
```

```
}

/* Ailleurs ... pour récupérer la ville d'un club donné */

$c = Club::where('id' ,'=', 23)->first();
$ville = $c->ville()->first();
```

D'après le modèle conceptuel des données (figure ci-dessus), un Tweet est rédigé par un utilisateur et un utilisateur rédige plusieurs Tweets. En s'inspirant de l'exemple ci-dessus :

1. *Dans le modèle **`Tweet`**, coder la méthode **`author`** qui retourne l'auteur du Tweet ;*
2. *Dans le modèle **`User`**, coder la méthodes **`tweets`** qui retourne les Tweets rédigés par l'auteur ;*
3. *Tester le fonctionnement.*

Association *->*

Pour les associations *->*, le principe reste le même. Supposons qu'un club a plusieurs usagers (enregistrés dans une table usagers) et qu'un usager peut appartenir à plusieurs clubs. Cette association se réalise en introduisant une table supplémentaire (une table pivot nommée **`usagers_club`**). Dans ce cas on défini une méthode **`adherents()`** dans le modèle **`Club`** qui retourne une collection d'objets **`Usagers`**. Inversement une méthode **`clubs()`** dans le modèle **`Usagers`** qui retourne une collection de d'objets **`Club`**.

Ici aussi *Eloquent* fourni une méthode pour réaliser cette association, la méthode **`belongsToMany()`**.

```
<?php

/* dans le modèle Club */

public function adherents() {
       return $this->belongsToMany('Usagers', 'usagers_clubs', 'club_id', 'usagers_id');

       /* 'Usagers'        : le nom de la classe du model lié */
       /* 'usagers_clubs ' : le nom de la table pivot */
       /* 'club_id'        : la clé étrangère de ce modèle dans la table pivot */
       /* 'usagers_id'     : la clé étrangère du modèle lié dans la table pivot */
}

/* Ailleurs ... pour récupérer les usagers d'un un club donné */

$c = Club::where('id' ,'=', 23)->first();
$liste_adherents = $c->adherents()->get();
```

```
<?php

/* dans le modèle Usagers */

public function clubs() {
       return $this->belongsToMany('Club', 'usagers_clubs', 'usagers_id', 'club_id');

       /* 'Clubs'          : le nom de la classe du model lié */
       /* 'usagers_clubs ' : le nom de la table pivot */

       /* 'usagers_id'     : la clé étrangère de ce modèle dans la table pivot */
       /* 'club_id'        : la clé étrangère du modèle lié dans la table pivot */
}

/* Ailleurs ... pour récupérer les clubs pour un usager donné */

$u = Usagers::where('id' ,'=', 45)->first();
$liste_clubs = $u->clubs()->get();
```

Attention : noter les symétries dans l'ordre des paramètres de la méthode **`belongsToMany`**.

Ici aussi d'après le modèle conceptuel des données, les Tweets sont appréciés (*likés*) par plusieurs utilisateurs et plusieurs utilisateurs peuvent apprécier un Tweets.

De plus, les utilisateurs suivent d'autres utilisateurs, et peuvent être suivit par plusieur utilisateurs.

En s'inspirant de l'exemple ci-dessus :

1. *Dans le modèle **`Tweet`**, coder la méthode **`likedBy`** qui retourne les utilisateurs qui ont appréciés le Tweet ;*

2. *Dans le modèle* ***`User`****, coder la méthode* ***`liked`*** *qui retourne les Tweets appréciés par l'utilisateur ;*
3. *Dans le modèle* ***`User`****, coder la méthode* ***`followedBy`*** *qui retourne les utilisateurs qui suivent l'auteur ;*
4. *Dans le modèle* ***`User`****, coder la méthodes* ***`follows`*** *qui retourne les utilisateurs suivis par l'auteur ;*
5. *Tester le fonctionnement.*

Votre Progrès

Avancement :

A ce stade vous devez avoir les fichiers suivants :

```
Tweeter
├── composer.json                          [Nouveau]
├── composer.lock                          [Nouveau]
├── conf
│   ├── config.ini                         [Nouveau]
│   └── tweeter.sql                        [Nouveau]
├── html
├── main.php
├── src
│   ├── mf
│   │   └── utils
│   │       ├── AbstractClassLoader.php
│   │       ├── AbstractHttpRequest.php
│   │       ├── ClassLoader.php
│   │       └── HttpRequest.php
│   └── tweeterapp
│       └── model
│           ├── Follow.php                 [Nouveau]
│           ├── Like.php                   [Nouveau]
│           ├── Tweet.php                  [Nouveau]
│           └── User.php                   [Nouveau]
├── tests
│   ├── ClassLoaderTest.php
│   ├── dummy
│   │   └── classname
│   │       └── Name.php
│   └── HttpRequestTest.php
└── vendor                                 [Nouveau]
```