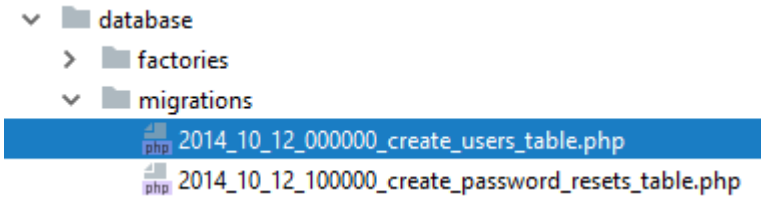
Créer une application avec Laravel 5.5 – Les données

Dans ce chapitre nous allons nous intéresser aux données et construire le schéma de la base pour la gestion de notre galerie photos. Laravel est équipé d'un efficace système de migrations couplé à un constructeur de schéma tout aussi efficace. D'autre part on dispose aussi d'un système de population (seeder) qui permet de remplir facilement les tables. On complétera ça en créant les modèles avec leurs relations.

Les utilisateurs

Dans l'installation de base on a déjà une migration pour les utilisateurs :



Avec ce code :

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
```

```
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Comme on doit distinguer les utilisateurs de base, qui peuvent juste ajouter et supprimer leurs photos et les administrateurs qui ont tous les droits il nous faut ajouter quelque chose.

Si on avait plus de distinctions à faire on ferait appel à l'un des nombreux packages qui existent comme bouncer, laratrust, ou laravel-permission. Mais dans notre cas ça serait vraiment excessif et on va se contenter d'ajouter une colonne de type **ENUM** :

```
$table->enum('role', ['user', 'admin'])->default('user');
```

On a donc les deux rôles : **user** et **admin** et par défaut c'est **user** qui est affecté.

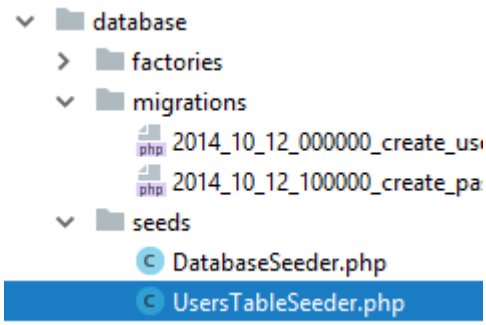
On complétera dans un chapitre ultérieur pour le profil mais pour le moment on va se contenter de ça.

Pour notre application on va aussi créer deux utilisateurs par défaut. Quand on regarde dans le fichier **database/seeds/DatabaseSeeder.php** on trouve ce code :

```
public function run()
{
    // $this->call(UsersTableSeeder::class);
}
```

On va dé-commenter la ligne et créer la classe **UsersTableSeeder** :

```
php artisan make:seeder UsersTableSeeder
```



On va changer le code pour celui-ci :

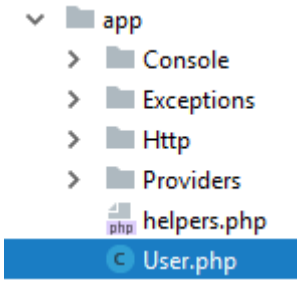
```
<?php

use Illuminate\Database\Seeder;
use App\Models\User;

class UsersTableSeeder extends Seeder
{

    public function run()
    {
        User::create([
            'name' => 'Durand',
            'email' => 'durand@chezlui.fr',
            'role' => 'admin',
            'password' => bcrypt('admin'),
        ]);

        User::create([
            'name' => 'Dupont',
            'email' => 'dupont@chezlui.fr',
            'password' => bcrypt('user'),
        ]);
    }
}
```

On aura ainsi un administrateur et un utilisateur.

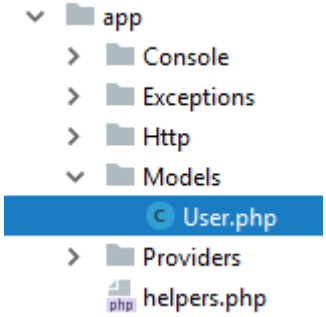
Les modèles

Comme on va créer de nouveaux modèles on va commencer par un peu réorganiser nos dossiers en prévoyant un dossier pour les modèles plutôt que de les empiler à la racine de l'application. Pour le

moment on a seulement **User** :



On crée le dossier **Models** et on déplace **User** dedans :



Dans le code on change l'espace de nom :

```
namespace App\Models;
```

Un truc plus vicieux facile à oublier est la configuration de l'authentification (**config/auth.php**) :

```
'providers' => [
    'users' => [
        'driver' => 'eloquent',
        'model' => App\Models\User::class,
    ],
],
```

Les catégories

Voyons ce dont nous aurons besoin pour la tables des catégories :

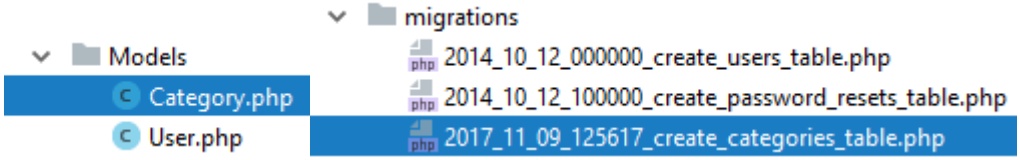
- un nom
- un slug

*Le slug est une version transformée du nom pour être intégrée dans une url. Si par exemple le nom est **Les Maisons** il est évident*

*qu'on ne peut pas intégrer ça directement dans une url. On va donc transformer le nom en **les-maisons** par exemple et là ça ira.*

Pour être efficace on va créer le modèle en même temps que la migration :

```
php artisan make:model Models\Category --migration
```



Dans le code de la migration changez ainsi la fonction **up** :

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name')->unique();
        $table->string('slug')->unique();
        $table->timestamps();
    });
}
```

On demande que les deux colonnes **name** et **slug** comportent des valeurs uniques, ce qui est logique.

Le modèle

Pour le modèle pour le moment on va juste ajouter les deux colonnes pour l'assignement de masse :

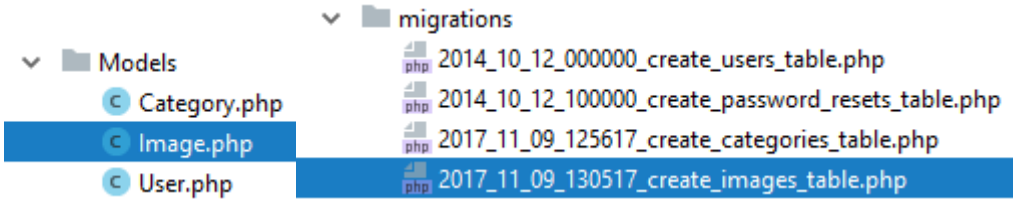
```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    protected $fillable = [
        'name', 'slug',
    ];
```

}

Les images

Il ne nous reste plus qu'à nous occuper des images, là aussi nous avons besoin du modèle et de la migration :

```
php artisan make:model Models\Image --migration
```



Pour les images on va avoir besoin :

- un nom (pas le nom de la photo mais celui du fichier !)
- une description optionnelle
- une clé étrangère pour les utilisateurs
- une clé étrangère pour les catégories

Ce qui donne cette migration :

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateImagesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('images', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('category_id')->unsigned();
            $table->integer('user_id')->unsigned();
```

```
            $table->string('name');
            $table->string('description')->nullable();
$table->foreign('category_id')->references('id')->on('categories')
->onDelete('cascade');
$table->foreign('user_id')->references('id')->on('users')->onDelet
e('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('images', function(Blueprint $table) {
            $table->dropForeign('images_category_id_foreign');
        });

        Schema::table('images', function(Blueprint $table) {
            $table->dropForeign('images_user_id_foreign');
        });

        Schema::dropIfExists('images');
    }
}
```

Remarquez que la colonne **description** est **nullable** puisqu'on la veut optionnelle.

D'autre part on fait le choix de la suppression en cascade, donc en cas de suppression d'un utilisateur ou d'un catégorie les images correspondantes seront aussi supprimées.

Pour le modèle pour le moment on va rien changer.

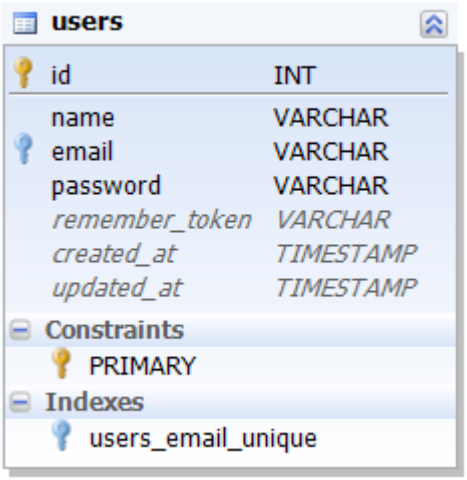
On migre !

On est maintenant prêts pour faire nos migrations :

```
php artisan migrate:fresh --seed
```

Table
categories
images
migrations
password_resets
users
5 tables

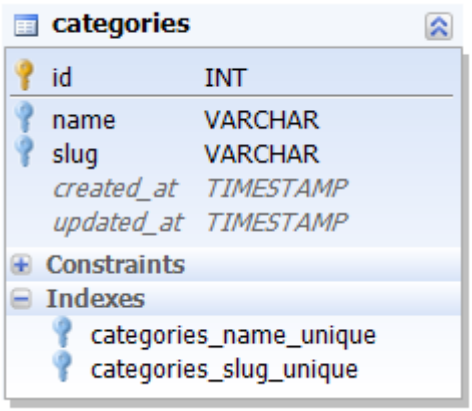
Vérifiez que tout est correct. Pour la table **users** :



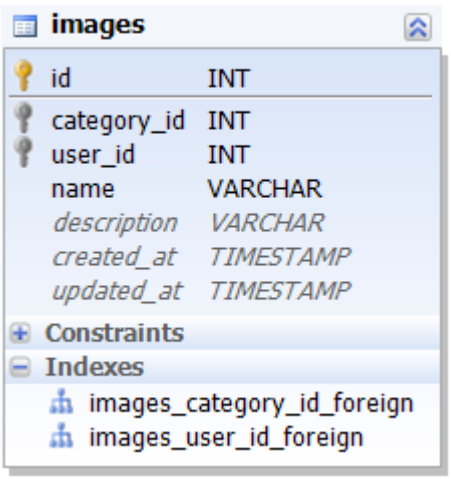
Avec les deux utilisateurs prévus :

id BIGI...	name VARCHAR(255)	email VARCHAR(255)	role ENUM	password VARCHAR(255)
1	Durand	durand@chezlui.fr	admin	$2y$10$mG2kdlNfjN0yyb.3WvnaKeXYUwJTl3nBr5RELjKGgssxYuW1pI6nG
2	Dupont	dupont@chezlui.fr	user	$2y$10$bJQBdaI.uba7LzRT.orseORDmg8s71ntGlhgX2gXUY81ps5Qwoxx.

La table **categories** :



Et enfin la table **images** :



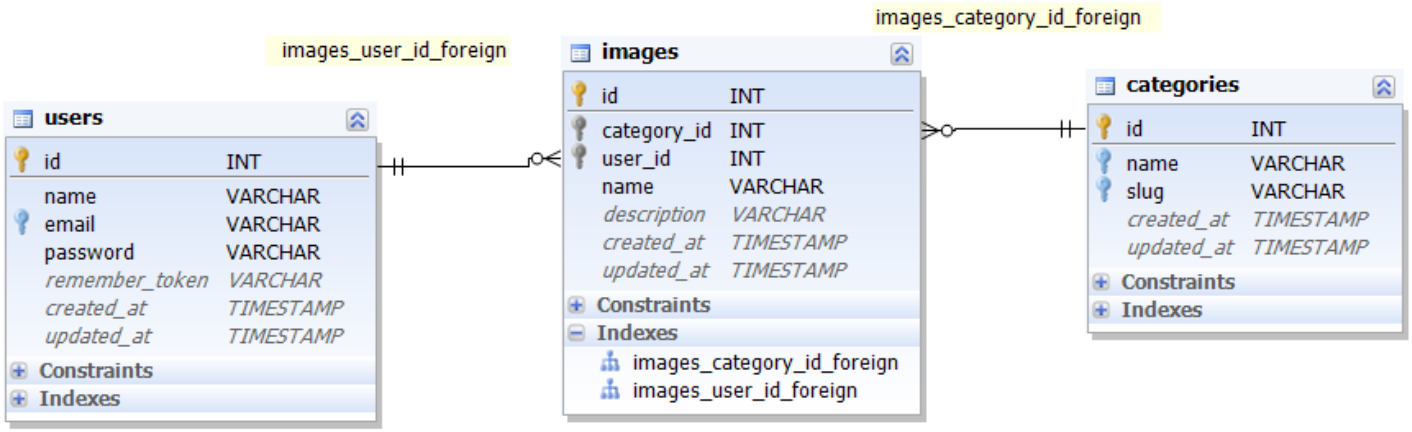
Les relations

On va établir ces relations :

- User **hasmany** Image
- Category **hasMany** Image

Et évidement les réciproques.

Voici le schéma de la base :



User

```
public function images()
{
    return $this->hasMany(Image::class);
}
```

Category

```
public function images()
{
    return $this->hasMany(Image::class);
}
```

Image

```
public function category()
{
    return $this->belongsTo(Category::class);
}

public function user()
{
    return $this->belongsTo(User::class);
}
```

Conclusion

Dans ce chapitre on a :

- créé toutes les migrations pour la galerie
- créé les modèles avec leurs relations
- créé un dossier spécifique pour les modèles
- créé deux utilisateurs dont un administrateur

Pour vous simplifier la vie vous pouvez charger le projet dans son état à l'issue de ce chapitre.