LARAVEL

Module 07

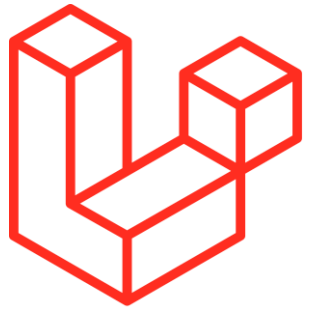
Eloquent, Migrations et Seeders

Au programme dans ce module

- Chapitre 1 : Configurations et .env
  - Con1iguration de la BDD
  - Place à la pratique
- Chapitre 2 : Les Migrations BDD
  - Fichiers de Migrations
  - Les commandes de Migrations
  - Configuration de la BDD avec Laravel
  - Place à la pratique
- Chapitre 3 : Qu'est ce que Laravel Eloquent ?
  - Les ORM
  - Laravel Eloquent
  - Les Modèles
  - Un coup de Collection
  - Place à la pratique
- Chapitre 4 : Les Seeders
  - Remplir vos tables
  - Place à la pratique
- Chapitre 5 : Pratiquons plus

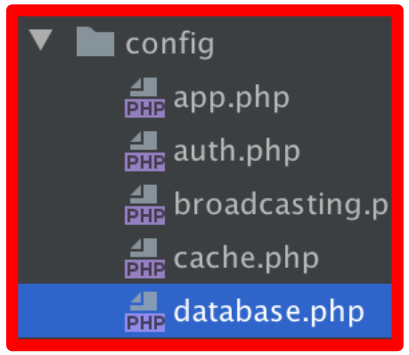
Configurations et .env

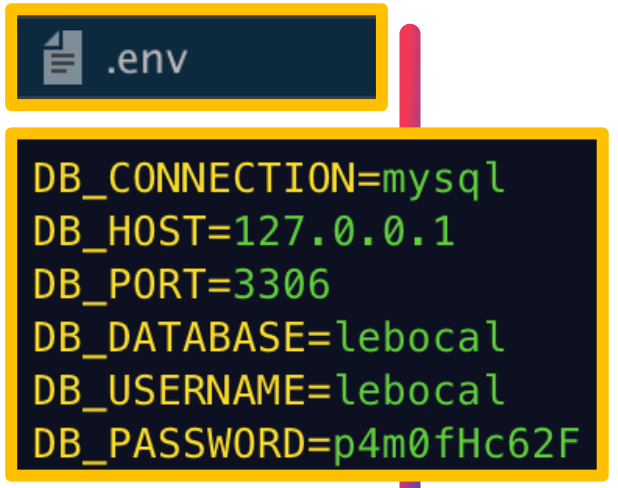
Chapitre 1



Configuration de la BDD

```php
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```





- « database.php » est le fichier de config des BDD pour Laravel
- .env est le fichier de variable d'environnement propre à l'application et plus particulièrement à l'emplacement d'exécution (serveur, pc, …).

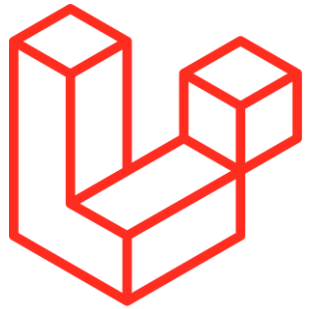
Place à la pratique

- Créez une base de données (BDD) avec PhpMyAdmin ou l'outil de votre choix.
- Créez un nouveau compte sur votre outil de gestion de BDD (ou à défaut, utilisez en un qui dispose des droits admin).
- Mettre à jour la configuration BDD de Laravel dans le « .env ».
- Testez votre configuration avec la commande :

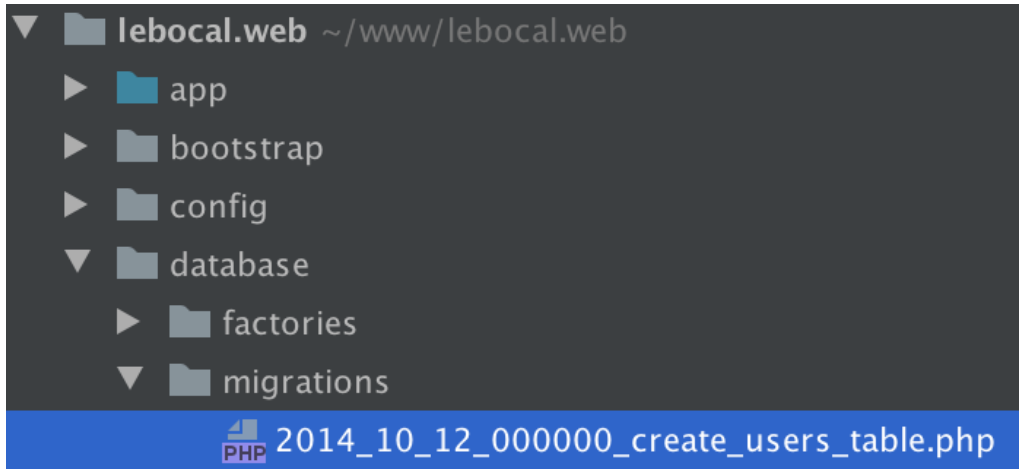
```
#> php artisan migrate:status
```

Les Migrations BDD

Chapitre 2



Les fichiers de Migrations




```
#> php artisan make:migration create_users_table
```

- Contient le schéma d'une table BDD
- 1 table = 1 fichier de Migration
- 1 update du schéma = 1 fichier de Migration
- Exécute la bonne commande selon le type de BDD
- 2 méthodes indispensable :
  - up() pour lancer la migration
  - down() pour inverser la migration

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

class CreateUsersTable extends Migration
{
    // Run the migrations.
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken(); // Pour les utilisateurs
            $table->timestamps(); // created_at + updated_at ()
        });
    }

    // Reverse the migrations.
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Les commandes de Migrations

```
#> php artisan make:migration create_users_table
```

→ Permet de créer un fichier de Migration

```
#> php artisan migrate
```

→ Lance toutes les nouveaux fichiers de migrations

```
#> php artisan migrate:reset
```

→ Rollback les migrations. ATTENTION, ceci lancera la méthode « down() » qui est généralement la suppression des tables.

```
#> php artisan migrate:refresh
```

→ « migrate:reset » + « migrate »

```
#> php artisan migrate:fresh
```

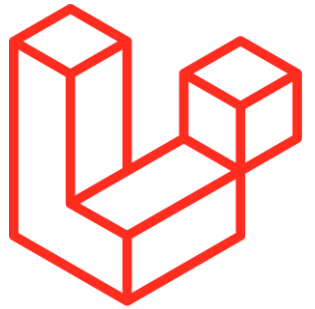
→ Supprime toutes les tables de la BDD et relance les migrations

Place à la pratique

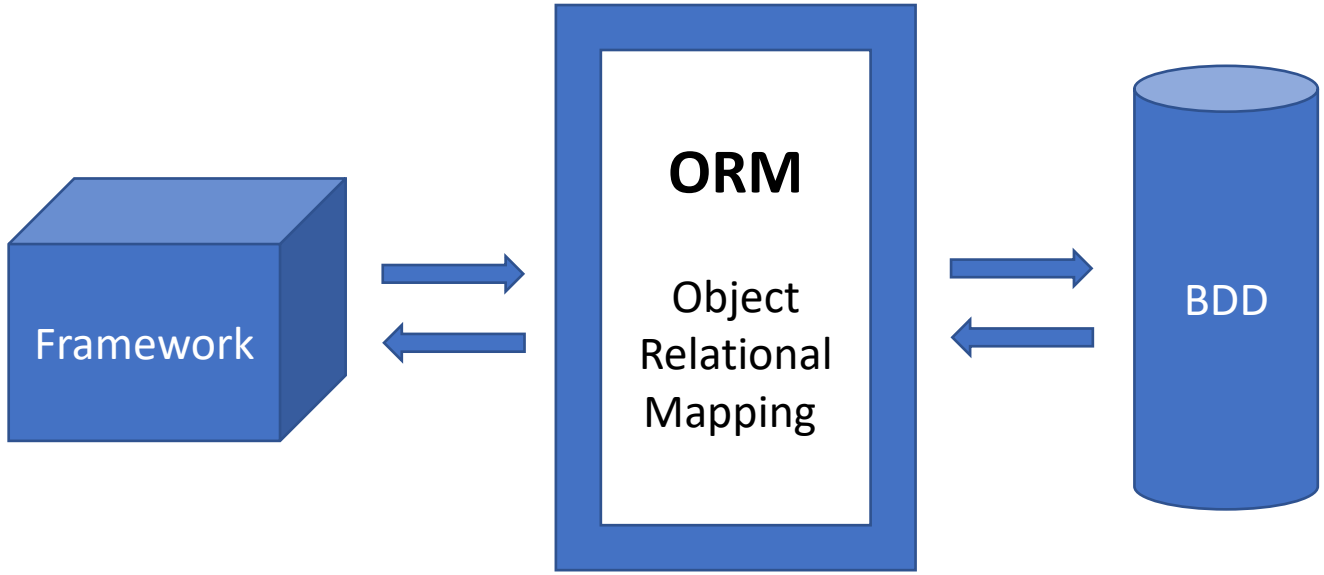
- Créez une migration pour une table « products » avec les champs que vous connaissez déjà.
- Ajoutez les colonnes suivantes dans le schéma de migration de « products » : **id** (comme pour la migration de user), **user_id** (int + nullable).
- En vous inspirant de la migration de user, utilisez la méthode « timestamps() » dans votre migration.
- Lancez les migrations. En cas de succès, essayez les autres commandes Artisan sur les migrations que vous venez de découvrir.

Qu’est ce que Laravel Eloquent ?

Chapitre 3



Les ORM



- Simplifie les requêtes BDD.
- Pour le SQL (nativement) et NoSQL (avec installation supplémentaire).
- Evite les répétitions de code SQL/NoSQL.
- Tant que l'ORM est compatible, il simplifie l'usage des groupes de types (string, int, float, boolean, …) en BDD pour les centraliser et les gérer automatiquement.
- Sécurité plus simple.

Laravel Eloquent

```php
use App\User;
```

```php
public function update(Request $request, $id)
{
    $user = User::select(['id', 'email'])->where('id', $id)->first();
    $user->name = $request->name;
    $user->password = bcrypt($request->password);
    $user->save();
}
```

```php
$user = User
    ::with('comments')
    ->where('id', $id)
    ->first();

foreach ($user->comments as &$c) {
    $c->title = strtoupper($c->title);
}

return view('user.show', [
    'user' => $user,
]);
```

- **Simple et rapide** à mettre en place
- **Simplifie les requêtes** d'écriture et de lecture, ainsi que les relations BDD
- **S'appuie sur les Collections** Laravel
- Génération et gestion de la pagination
- Compatible MySQL, PostgreSQL, SQLite, SQL Server et équivalent (MariaDB, …)
- Gestion simplifié des transactions
- **Eager Loading** pour facilité l'usage des relations

Les Modèles

```php
use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    // The table associated with the model.
    protected $table = 'hotels';

    // The primary key for the model.
    protected $primaryKey = 'id';

    // The "type" of the auto-incrementing ID.
    protected $keyType = 'string';

    // Indicates if the IDs are auto-incrementing.
    protected $incrementing = TRUE;

    // The attributes that are mass assignable.
    protected $fillable = [
        'user_id', 'title', 'message',
    ];

    // The attributes that should be hidden for arrays.
    protected $hidden = [
        //
    ];
}
```

- **1 table = 1 Model** (sauf les tables pivot dans les relations N,N).
- **Obligatoire** : « extends Model ».
- **Nom du Model au singulier !**
- Nom de table automatique (ou spécifiable).
- Plusieurs options de configuration comme le nom de la clé primaire, ou son type.
- Il faut choisir des colonnes autorisées à l'insertion de masse.
- Il faut choisir des colonnes à ne pas récupérer par défaut sans spécifier de « select ».

Un coup de Collection 1/2

```php
$users = User::all();

$usersArray = $users->toArray(); // Récupère les colonnes/valeurs dasn un tableau
$usersId = $users->pluck('email', 'id'); // Récupère uniquement un couple : id => email
                                         // Second paramètre optionnel
$users = $users->keyBy('email'); // Change les clés de la collection par les "email"
$ca = $users->sum('subscription_price'); // Fait la somme des "subscription_price"
$users = $users->filter(function ($user, $key) { // Tous les utilisateurs avec une
    return $user->subscription_price > 9.99;     // souscription supérieur à 9.99
});
```

- Un peu comme un nouveau type de variable créée par Laravel basé sur une impossante classe et les méthodes magiques offertes par PHP.
- Une Collection fonctionnant un peu comme un tableau PHP. Mise à part les fonctions de traitement des tableaux en PHP, une Collection se comporte aussi comme un tableau.
- Une centaine de méthodes de traitements et de filtres pour faire à peu près tout.
- Il existe des Collection encore plus poussée pour Eloquent ORM.

Un coup de Collection 2/2

all
average
avg
chunk
collapse
collect
combine
concat
contains
containsStrict
count
countBy
crossJoin
dd
diff
diffAssoc
diffKeys
dump
duplicates
duplicatesStrict
join
keyBy
keys
last
macro
make
map
mapInto
mapSpread
mapToGroups
mapWithKeys
max
median
merge
mergeRecursive
min
mode
nth
only
pad
sort
sortBy
sortByDesc
sortKeys
sortKeysDesc
splice
split
sum
take
tap
times
toArray
toJson
transform
union
unique
uniqueStrict
unless
unlessEmpty
unlessNotEmpty
each
eachSpread
every
except
filter
first
firstWhere
flatMap
flatten
flip
forget
forPage
get
groupBy
has
implode
intersect
intersectByKeys
isEmpty
isNotEmpty
partition
pipe
pluck
pop
prepend
pull
push
put
random
reduce
reject
replace
replaceRecursive
reverse
search
shift
shuffle
skip
slice
some
unwrap
values
when
whenEmpty
whenNotEmpty
where
whereStrict
whereBetween
whereIn
whereInStrict
whereInstanceOf
whereNotBetween
whereNotIn
whereNotInStrict
whereNotNull
whereNull
wrap
zip

Place à la pratique

- Découverte des Collections :

- Créez un tableau de 3 nombres différents (exemple : [740, 88.5, -35])
- Utilisez le helper « **collect()** » pour transformer votre tableau en Collection (Voir la documentation).
- Utilisez les méthodes des Collections suivante : « **sum()** », « **average()** » et « **sort()** » depuis votre collection (voir cours).

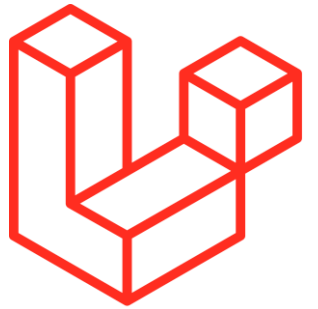
Place à la pratique

- Découverte des Models:

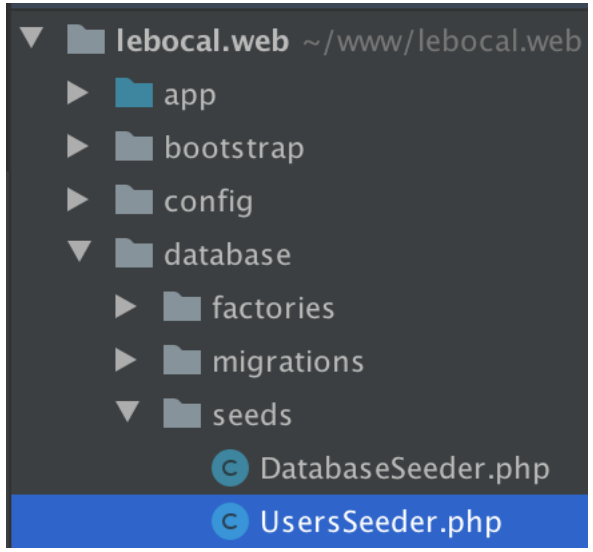
- Créer un Model « **Pokemon** ».
- Finir la création d'un produit en utilisant le modèle Pokemon et votre contrôleur Pokemons. Et testez cette création, du formulaire à la base de données.
- Modifiez la page **pokemons.index** qui doit maintenant afficher la liste des pokemons.
- Modifiez la page **pokemons.show** qui doit maintenant afficher les données d'un seul pokemon (par son id).
- Si ce n'est pas déjà fait, créez un formulaire pour la mise à jour des données d'un produit (pokemons.edit).
- Maintenant, finissez les méthodes de votre contrôleur Pokemon: update() et delete()/destroy().
- Puis ajoutez les liens UD (du CRUD) dans votre vue « pokemons.index ». Et testez les.

Les Seeders BDD

Chapitre 4



Remplir vos tables




```
#> php artisan make:seeder UsersTableSeeder
```

- Optionnel, il sert uniquement à insérer des données dans des tables de la BDD lors de l'installation ou de la mise à jour de l'application.
- Souvent, utilisé en même temps qu'une migration.
- Utile pour créer des comptes par défaut ou des données de configuration BDD essentielles à l'app.

```php
use Illuminate\Database\Seeder;

class UsersTableSeeder extends Seeder
{
    // Run the database seeds.
    public function run()
    {
        $now = now();

        DB::table('users')->insert([
            'name' => Str::random(10),
            'email' => Str::random(10).'@gmail.com',
            'password' => bcrypt('password'),
            'created_at' => $now,
            'updated_at' => $now,
        ]);
    }
}
```

DatabaseSeeder.php

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    // Seed the application's database.
    public function run()
    {
        $this->call(UsersTableSeeder::class);
    }
}
```

Les commandes des Seeders

```
#> php artisan make:seeder UsersTableSeeder
```

→ Permet de créer un fichier de Seeder

```
#> php artisan db:seed
```

→ Lance toutes les seeders référencés dans « /database/seeds/DatabaseSeeder.php »

```
#> php artisan db:seed --class=UsersTableSeeder
```

→ Lance un seeder précis

```
#> php artisan migrate --seed
```

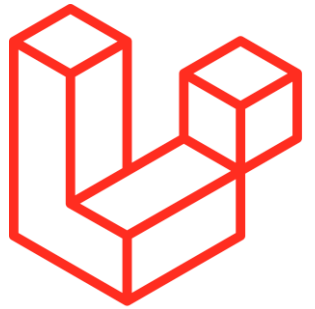
→ L'option ---seed peut être utiliser avec les commandes de migrations

Place à la pratique

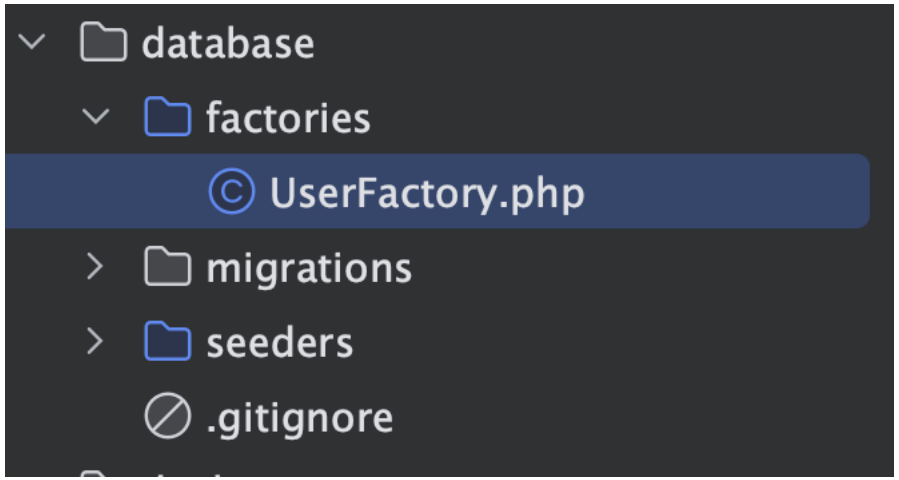
- Créez un seeder pour la table des pokemons !
- Trouvez et ajoutez 3 pokemons par défaut qui seront toujours présent dans l'application.
- Testez les commandes des migrations (pour reconstruire une base de données clean) puis des seeders (pour remplir avec de veritable données).

Factories

Chapitre 5



Factory power



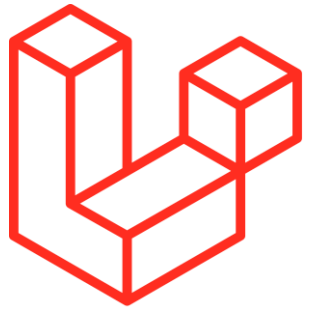

```php
$users = UserFactory::new()->count( count: 100);
```

→ Run une factory de 100 users en BDD et récupération dans une collection

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    public function definition(): array
    {
        return [
            'name' => fake()->name(),
            'email' => fake()->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' => Hash::make('password'),
            'remember_token' => Str::random(10),
        ];
    }
}
```

Pratiquons plus

Chapitre 5



Place à la pratique

- Ajoutez des liens entre les modèles User et Pokemon.
- Trouvez comment ajouter une relation **1,1 (One to One)** entre le Model **Pokemon** et le Model **User**.
- Trouvez comment ajouter une relation **1,N (One to Many)** entre le Model **User** et le Model **Pokemon**.
- (Les relations entre les Models permettent d'utiliser le **Eager Loading** (voir doc: https://laravel.com/docs/10.x/eloquent-relationships#eager-loading).

Fin du module