Cours Laravel 6 – les données – jouer avec Eloquent

laravel.sillo.org/cours-laravel-6-les-donnees-jouer-avec-eloquent/

bestmomo

August 31, 2019

On a commencé à voir Eloquent et les modèles dans le premier chapitre de cette partie du cours, mais on s'est limité pour le moment à créer un enregistrement. Dans ce chapitre on va aller plus loin en explorant les possibilités d'Eloquent, couplé à un puissant générateur de requêtes, pour manipuler les données.

Pour effectuer les manipulations de ce chapitre vous aurez besoin d'une installation fraîche de Laravel complétée avec les éléments de l'authentification comme on l'a vu dans le chapitre correspondant. Les migrations devront également être effectuées pour qu'on puisse utiliser la base de données.

Tinker

Artisan est plein de possibilités, l'une d'elle est un outil très pratique, un **REPL** (Read–Eval–Print Loop). C'est un outil qui permet d'entrer des expressions, de les faire évaluer et d'avoir le résultat à l'affichage et son nom est **Tinker**. Ce REPL est boosté par le package psysh. Il y a un bon article de présentation en anglais ici.

Pour lancer cet outil il y a une commande d'artisan :

```
php artisan tinker
```

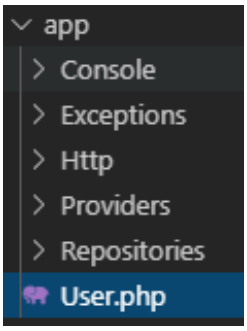
On va se servir de cet outil pour interagir avec la base de données dans ce chapitre.

Créer un enregistrement

Méthode **create**

On a déjà vu comment créer un enregistrement avec Eloquent. Comme a priori votre base de données est vide avec une installation fraiche de Laravel on va commencer par ajouter un utilisateur dans la table **users** à l'aide du modèle **User**. Pour mémoire ce modèle est le seul présent au départ :

Dans Tinker entrez cette expression :




```
User::create(['name' => 'Durand', 'email' => 'durand@chezlui.fr', 'password' => 'pass'])
```

```
>>> User::create(['name' => 'Durand', 'email' => 'durand@chezlui.fr', 'password' => 'pass'])
[!] Aliasing 'User' to 'App\User' for this Tinker session.
=> App\User {#2983
     name: "Durand",
     email: "durand@chezlui.fr",
     updated_at: "2019-08-31 12:23:36",
     created_at: "2019-08-31 12:23:36",
     id: 4,
   }
>>>
```

*Remarquez qu'on n'a pas besoin d'entrer l'espace de nom complet **App\User**, Tinker crée automatiquement un alias.*

La méthode **create** renvoie le modèle avec tous les attributs créés comme les dates et l'id. Tinker nous les affiche gentiment. On retrouve évidemment notre enregistrement dans la table **users** :

name VARCHAR(255)	email VARCHAR(255)	email_verified_at TIMESTAMP	password VARCHAR(255)	remember_token VARCHAR(100)	created_at TIMESTAMP	updated_at TIMESTAMP
Durand	durand@chezlui.fr	*(null)*	pass	*(null)*	31/08/2019 12:23:36	31/08/2019 12:23:36

*Pour mémoire cette méthode **create** est un assignement de masse et implique les précautions que nous avons déjà vues.*

Méthode **save**

Une autre façon de créer un enregistrement avec Eloquent et de commencer par créer un modèle vide, renseigner les attributs, puis utiliser la méthode **save** pour enregistrer dans la table.

Entrez cette suite d'expressions :

```
$user = new User
$user->name = 'Dupont
$user->email = 'dupont@chezlui.fr'
$user->password = 'pass'
$user->save()
```

On commence par créer le modèle, puis on renseigne ses attributs, enfin on enregistre dans la base avec **save**.

```
>>> $user = new User
=> App\User {#749}
>>> $user->name = 'Dupont'
=> "Dupont"
>>> $user->email = 'dupont@chezlui.fr'
=> "dupont@chezlui.fr"
>>> $user->password = 'pass'
=> "pass"
>>> $user->save()
=> true
```

Modifier un enregistrement

Voyons maintenant comment modifier un enregistrement existant. Il y a aussi deux façons de procéder.

Méthode **update**

Entrez cette expression dans Tinker :

```
User::find(1)->update(['email' => 'durand@cheznous.fr'])
```

```
>>> User::find(1)->update(['email' => 'durand@cheznous.fr']);
=> true
```

On commence par récupérer le premier enregistrement de la table avec **find(1)**, ça a pour effet de créer un modèle avec les attributs renseignés pour Durand. Ensuite on utilise la méthode **update** sur ce modèle en précisant le nom et la valeur de chaque attribut qu'on veut modifier dans un tableau. ici on a changé l'adresse email pour Durand.

Avec cette méthode update on a encore le problème de l'assignement de masse.

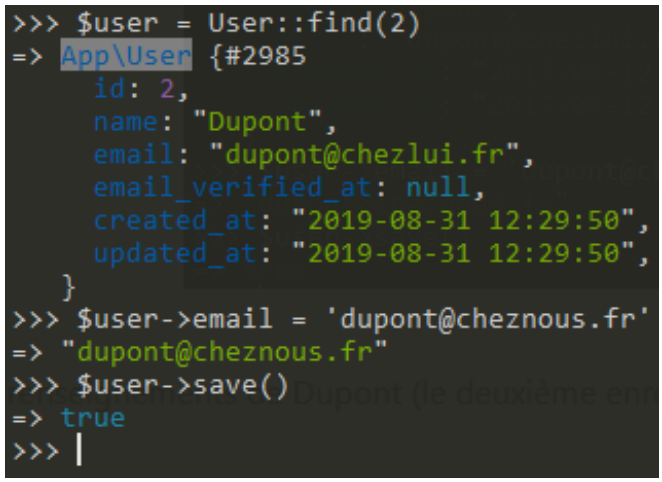
Méthode **save**

On peut aussi utiliser la méthode **save** pour faire une modification. Entrez ces expressions dans Tinker :

```
$user = User::find(2)
$user->email = 'dupont@cheznous.fr'
$user->save()
```

Avec **find(2)** on crée un modèle avec les renseignements de Dupont (le deuxième enregistrement de la table), ensuite on change l'attribut **email**, enfin on enregistre dans la base.

On peut constater les modifications dans la table :



id UNSIGNED BIGINT(20)	name VARCHAR(255)	email VARCHAR(255)	email_verified_at TIMESTAMP	password VARCHAR(255)	remember_token VARCHAR(100)	created_at TIMESTAMP	updated_at TIMESTAMP
1	Durand	durand@chezn...	*(null)*	pass	*(null)*	31/08/2019 12:23:36	31/08/2019 12:27:55
2	Dupont	dupont@chezn...	*(null)*	pass	*(null)*	31/08/2019 12:29:50	31/08/2019 12:30:59

*Remarquez que les colonnes **created_at** et **updated_at** n'ont maintenant plus les mêmes valeurs puisqu'on a fait des modifications.*

Supprimer un enregistrement

Après la création et la modification vient tout naturellement la suppression avec la méthode **delete**. Si vous entrez cette expression dans Tinker :

```
User::find(2)->delete()
```

Vous supprimez l'utilisateur qui a l'index 2.

La suppression est définitive. Il existe aussi une possibilité de suppression provisoire (**soft delete**) mais il faut modifier le modèle en ajoutant un trait et une propriété. Cette partie de la documentation traite de ce sujet que je ne pense pas développer dans ce cours.

Trouver des enregistrements

Rafraichissez la table (pas dans Tinker cette fois) :

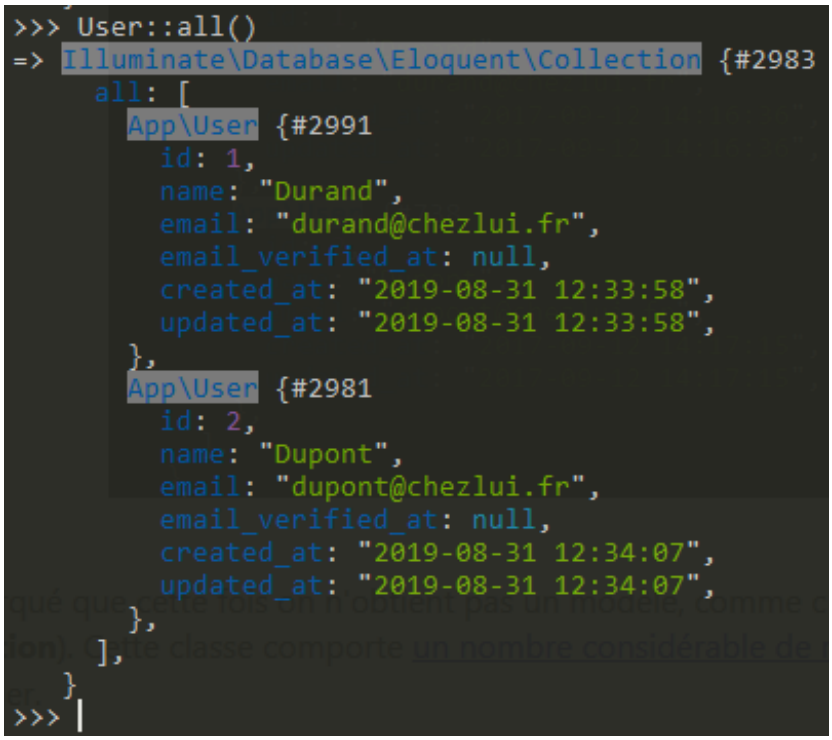
```
php artisan migrate:refresh
```

Et recréez avec Tinker les deux utilisateurs :

```
User::create(['name' => 'Durand', 'email' => 'durand@chezlui.fr', 'password' => 'pass'])
User::create(['name' => 'Dupont', 'email' => 'dupont@chezlui.fr', 'password' => 'pass'])
```

Tous les enregistrements

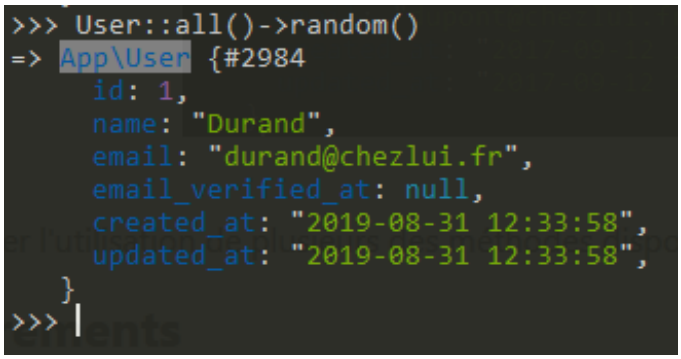
On a vu déjà la méthode **find** pour trouver un enregistrement à partir de son identifiant (on peut aussi passer plusieurs identifiants dans un tableau). On peut aussi tous les récupérer avec la méthode **all** :



Si vous êtes observateur vous avez remarqué que cette fois on n'obtient pas un modèle, comme c'était le cas avec **find** mais une collection (**Illuminate\Database\Eloquent\Collection**). Cette classe comporte un nombre considérable de méthodes, on peut donc effectuer de nombreux traitement à ce niveau et même les chaîner.

Par exemple ici j'extrais un modèle au hasard de la collection :

J'aurai l'occasion dans ce cours de montrer l'utilisation de plusieurs des méthodes disponibles.



Sélectionner des enregistrements

En général on ne veut pas tous les enregistrements mais juste certains qui correspondent à certains critères. Eloquent est couplé à un puissant générateur de requêtes SQL (**Query Builder**). On peut par exemple utiliser la méthode **where**. Entrez cette expression dans Tinker :

```
User::where('name', 'Dupont')->get()
```

```
>>> User::where('name', 'Dupont')->get()
=> Illuminate\Database\Eloquent\Collection {#2991
     all: [
       App\User {#2990
         id: 2,
         name: "Dupont",
         email: "dupont@chezlui.fr",
         email_verified_at: null,
         created_at: "2019-08-31 12:34:07",
         updated_at: "2019-08-31 12:34:07",
       },
     ],
   }
```

On récupère une collection qui ne contient que le modèle qui correspond à Dupont.

Ajoutez cet utilisateur :

```
User::create(['name' => 'Martin', 'email' => 'martin@chezlui.fr', 'password' => 'pass'])
```

On a maintenant 3 utilisateurs :

Maintenant entrez cette expression :

```
User::where('name', '<', 'e')->get()
```

id UNSIGNED BIGINT(20)	name VARCHAR(255)
1	Durand
2	Dupont
3	Martin

```
>>> User::where('name', '<', 'e')->get()
=> Illuminate\Database\Eloquent\Collection {#2998
     all: [
       App\User {#2986
         id: 1,
         name: "Durand",
         email: "durand@chezlui.fr",
         email_verified_at: null,
         created_at: "2019-08-31 12:33:58",
         updated_at: "2019-08-31 12:33:58",
       },
       App\User {#2989
         id: 2,
         name: "Dupont",
         email: "dupont@chezlui.fr",
         email_verified_at: null,
         created_at: "2019-08-31 12:34:07",
         updated_at: "2019-08-31 12:34:07",
       },
     ],
   }
```

Cette fois on a sélectionné à partir des noms dont la première lettre doit être avant « e » dans l'alphabet et on obtient une collection avec deux modèles.

On dispose de la méthode **first** si on veut récupérer juste le premier :

```
>>> User::where('name', '<', 'e')->get()->first()
=> App\User {#2991
     id: 1,
     name: "Durand",
     email: "durand@chezlui.fr",
     email_verified_at: null,
     created_at: "2019-08-31 12:33:58",
     updated_at: "2019-08-31 12:33:58",
   },
```

On peut aussi limiter les colonnes retournées avec la méthode **select** :

```
>>> User::select('name')->where('name', '<', 'e')->get()
=> Illuminate\Database\Eloquent\Collection {#2992
     all: [
       App\User {#2998
         name: "Durand",
       },
       App\User {#2999
         name: "Dupont",
       },
     ],
   },
```

Ici on ne retourne que les noms.

Souvent on a envie de connaître la requête générée, pour la voir il suffit d'utiliser la méthode **toSql** :

On dispose avec le générateur de requête de riches possibilités que nous verrons progressivement, par exemple on peut facilement trier les enregistrements avec la méthode **orderBy** :

```
>>> User::select('name')->toSql()
=> "select `name` from `users`"
```

```
>>> User::select('name')->orderBy('name', 'desc')->get()
=> Illuminate\Database\Eloquent\Collection {#2975
     all: [
       App\User {#2997
         name: "Martin",
       },
       App\User {#2984
         name: "Durand",
       },
       App\User {#2993
         name: "Dupont",
       },
     ],
   },
```

Souvent il existe un syntaxe plus simple, par exemple on peut obtenir le même résultat avec **latest** :

```
User::select('name')->latest('name')->get()
```

Trouver ou créer

Que se passe-t-il si l'enregistrement qu'on veut ne se trouve pas dans la table ? Faisons un essai :

```
>>> User::find(4)
=> null
```

On a **null** en retour et on peut gérer cette valeur. Mais des fois on se trouve dans une situation où on veut récupérer un enregistrement ou le créer s'il n'existe pas. On peut le faire en deux étapes mais on peut aussi le faire en une seule action avec la méthode **firstOrCreate** :

```
>>> User::firstOrCreate(['name' => 'Martin', 'email' => 'martin@chezlui.fr', 'password' => 'pass'])
=> App\User {#2998
     id: 3,
     name: "Martin",
     email: "martin@chezlui.fr",
     email_verified_at: null,
     created_at: "2019-08-31 12:37:24",
     updated_at: "2019-08-31 12:37:24",
   }
```

Ici comme l'enregistrement existe il est retourné normalement mais dans le cas suivant il n'existe pas et donc il est créé :

```
>>> User::firstOrCreate(['name' => 'Lulu', 'email' => 'lulu@chezlui.fr', 'password' => 'pass'])
=> App\User {#2981
     name: "Lulu",
     email: "lulu@chezlui.fr",
     updated_at: "2019-08-31 12:44:27",
     created_at: "2019-08-31 12:44:27",
     id: 5,
   }
```

On se retrouve donc avec 4 enregistrements dans la table :

id UNSIGNED BIGINT(20)	name VARCHAR(255)	email VARCHAR(255)
1	Durand	durand@chezlui.fr
2	Dupont	dupont@chezlui.fr
3	Martin	martin@chezlui.fr
5	Lulu	lulu@chezlui.fr

Il existe aussi la méthode **firstOrNew**, si l'enregistrement existe il est retourné comme avec **firstOrCreate**, par contre s'il n'existe pas il est juste créé une instance du modèle mais aucune action dans la base n'est réalisée.

De la même manière il existe la méthode **updateOrCreate** basée sur le même principe.

En résumé

- On peut créer des enregistrements avec Eloquent avec la méthode **create**.
- On peut modifier des enregistrements avec Eloquent avec la méthode **update**.
- On peut créer ou modifier des enregistrements avec Eloquent avec la méthode **save**.
- Le résultat d'une requête générée avec Eloquent est soit un modèle soit une collection.
- Eloquent est associé à un puissant générateur de requêtes (Query Builder).
- Des méthodes spéciales permettent de faire plusieurs actions comme aller

chercher un enregistrement ou le créer s’il n’existe pas.