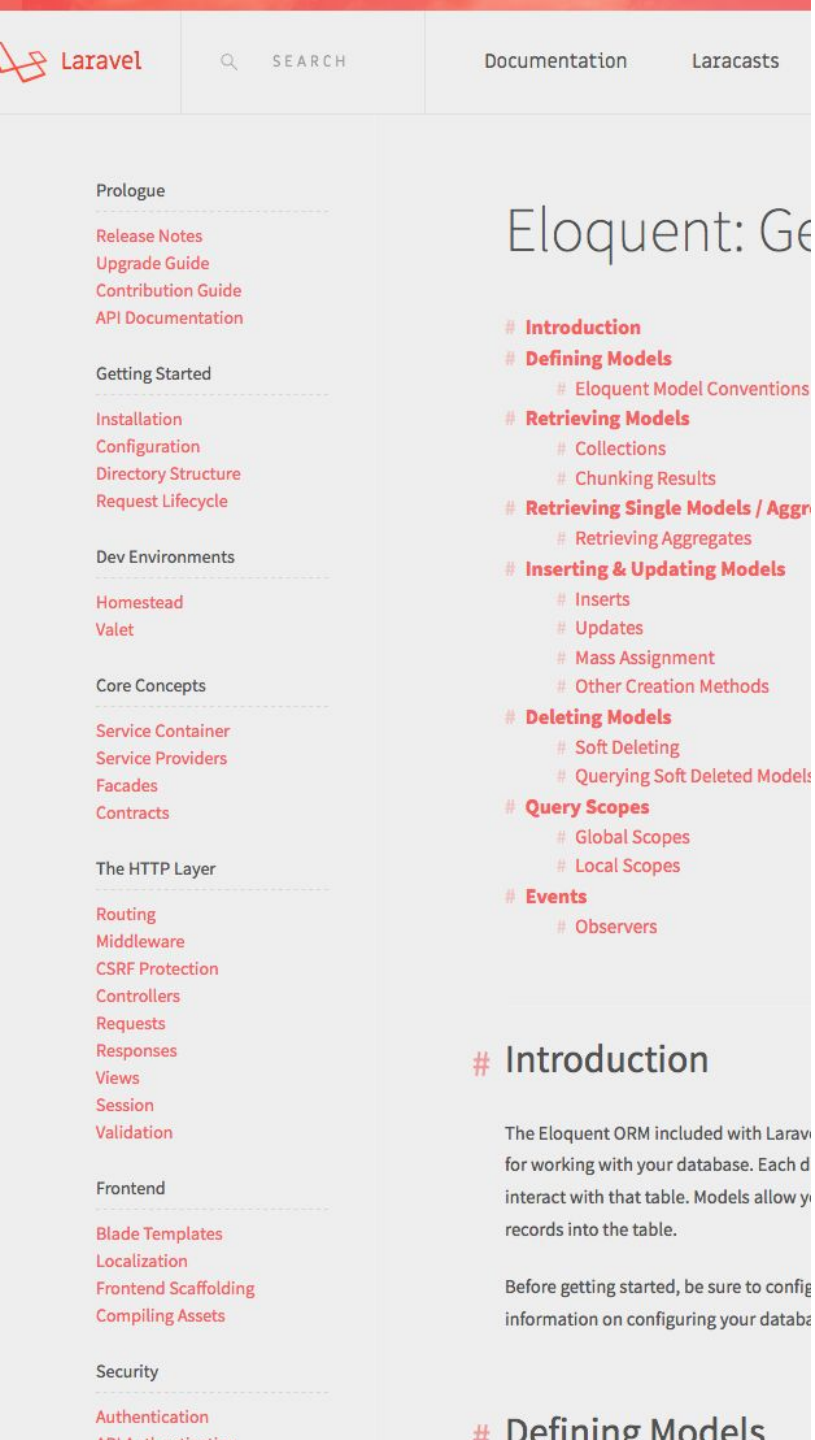


LARAVEL: ELOQUENT ORM

DISEÑO DE SISTEMAS SOFTWARE

Contenido

1. Eloquent ORM
2. Operaciones con modelos
3. Relaciones
4. Ejercicio

Eloquent ORM

- ORM (*Object-Relational mapping* o mapeo objeto-relacional) es una técnica de programación para convertir datos entre un lenguaje orientado a objetos y una BD relacional como motor de persistencia
- Laravel incluye su propio sistema de ORM llamado Eloquent
- Eloquent proporciona una manera elegante y fácil de interactuar con la BD a través de PHP
- Está basado en el patrón de acceso a datos **ActiveRecord**: los objetos de lógica de negocio son los encargados de acceder a su tabla correspondiente en la base de datos

Eloquent ORM

- Cada tabla en la BD debe tener su correspondiente modelo en la carpeta `app` (aunque se puede configurar para usar otro sitio)
- Para crear un nuevo modelo de datos podemos usar el comando de Artisan:

```
$ php artisan make:model Product
```

- Este comando creará el fichero `Product.php` en la carpeta `app`
- Si añadimos el parámetro `-m` creará también el archivo de migraciones

```
$ php artisan make:model Product -m
```

Eloquent ORM

- Al crear un nuevo modelo con Artisan se incluirá el contenido básico del mismo
- Por ejemplo, la clase o modelo `Product.php` sería:

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class Product extends Model // Heredamos de Model
{
    //
}
```

- Solamente con este código y sin escribir nada más podemos utilizarlo para realizar todo tipo de queries sobre la tabla `products`

Eloquent ORM

- Eloquent automáticamente enlaza el modelo con la tabla a partir del nombre de la clase, transformándolo al plural en minúsculas:
  - `User` → `users` // Usará el plural en inglés
  - Para cambiar el nombre usamos la propiedad `$table` del modelo
- Asume que la tabla tendrá una clave primaria llamada `id`
  - Para cambiar el nombre usamos la propiedad `$primaryKey`
- Además actualiza automáticamente los timestamps de la tabla (`updated_at` y `created_at`).
  - Para desactivarlo usamos la propiedad `$timestamps`

Eloquent ORM

- Ejemplo de personalización de un modelo

```php
class User extends Model
{
    protected $table = 'my_users';
    protected $primaryKey = 'my_id';
    public $timestamps = false;
}
```

- Más información en https://laravel.com/docs/6.x/eloquent

Laravel: Eloquent ORM

OPERACIONES CON MODELOS

Consultas

- Para obtener todas las filas de la tabla asociada al modelo `Product` utilizaremos el método `all()`:

```php
use App\Product;  // Indicamos su espacio de nombres!

$products = Product::all(); // select * from products

foreach( $products as $product ) {
        var_dump( $product->name );
}
```

- Este método nos devolverá un array de resultados, donde cada item del array es una instancia del modelo `Product`
- Esto nos permite acceder a los valores de cada elemento como si fuera un objeto: `$product->name`

Consultas

- Todos los métodos de **Query Builder** se pueden utilizar con Eloquent. Además Eloquent incorpora algunos más
- Buscar un elemento por su identificador (por defecto `id`):

```php
$product = Product::find(1);
var_dump($product->name);
```

- Si queremos que se lance una excepción cuando no se encuentre un modelo:

```php
$model = Product::findOrFail(1);
$model = Product::where('price', '>', 100.0)->firstOrFail();
```

- Esto nos permite capturar las excepciones y mostrar un error 404 cuando sucedan.

Consultas, ejemplos

```php
$products = Product::where('price', '>', 100.0)->take(10)->get();
foreach ($products as $product)
{
    var_dump($product->name);
}

// O para obtener el primer producto de la lista
$product = Product::where('price', '>', 100.0)->first();

$count = Product::where('price', '>', 100.0)->count();
$price = Product::max('price');
$price = Product::min('price');
$price = Product::avg('price');
$total = Product::sum('price');
```

Insertar datos

- Para añadir una entrada en la tabla de la base de datos asociada con un modelo tenemos que hacer:

```php
$product = new Product();
$product->name = 'WD 3TB';
$product->save();

// Dentro de un seeder podemos usar la sintaxis abreviada
$product = new Product(['name' => 'WD 3TB']);
$product->save();
```

- Para obtener el identificador asignado en la base de datos **después de guardar:**

```php
$insertedId = $product->id;
```

Actualizar datos

- Para actualizar un registro de un modelo solo tendremos que recuperar en primer lugar la instancia, modificarla y por último guardar los datos:

```
$product = Product::find(1);
$product->price = 157.35;
$product->save();
```

- Recuerda que también puedes usar `findOrFail`

Borrar datos

- Para borrar una instancia de un modelo en la base de datos usamos el método `delete()`:

```
$product = Product::find(1);
$product->delete();
```

- También podemos borrar un conjunto de resultados:

```
$affectedRows = Product::where('price','>',100.0)->delete();
```

Laravel: Eloquent ORM

RELACIONES

Relaciones

- Las relaciones se definen como métodos dentro de las clases
- Estos métodos devuelven los objetos relacionados en el otro extremo, p.ej. los posts relacionados con un usuario:

```
    $user->posts()->where('active', 1)->get();
```

Relaciones

- Para que las relaciones establecidas en el código funcionen, tienen que existir las claves ajenas correspondientes en la base de datos
- Las claves ajenas deben seguir el formato `<modelo>_id` (p.ej. `category_id`)
- Para utilizar nombres de tablas y/o columnas distintas existen versiones de los métodos expuestos a continuación que aceptan parámetros adicionales: https://laravel.com/docs/6.x/eloquent-relationships

Relaciones 1—1

- El objeto principal accede al relacionado con `hasOne()`
- El objeto relacionado que contiene la clave ajena accede al objeto principal con el método `belongsTo()`

```php
class User extends Model {
    public function phone() {
        return $this->hasOne('App\Phone');
    }
}

class Phone extends Model {
    public function user() {
        // Phone tiene la clave ajena 'user_id'
        return $this->belongsTo('App\User');
    }
}
```

Relaciones 1—1

- Para guardar objetos relacionados se usan los métodos `save()` (en el objeto principal) y `associate()` (en el relacionado)

```php
$user = new User(['name' => 'username1']);
$user->save();
$phone = new Phone(['number' => '987654321']);
$user->phone()->save($phone);

// las dos versiones son equivalentes

$user = new User(['name' => 'username2']);
$user->save();
$phone = new Phone(['number' => '123456789']);
$phone->user()->associate($user);
$phone->save();
```

Relaciones 1—*

- El objeto principal accede a los relacionados con `hasMany()`
- Los objetos relacionados que contienen la clave ajena acceden al objeto principal con el método `belongsTo()`

```
class Category extends Model {
    public function products() {
        return $this->hasMany('App\Product);
    }
}

class Product extends Model {
    public function category() {
        // Product tiene la clave ajena 'category_id'
        return $this->belongsTo('App\Category);
    }
}
```

Relaciones 1—*

- Para guardar objetos relacionados se usan los métodos `save()` o `saveMany()` (en el objeto principal) y `associate()` (en el relacionado)

```php
$category = new Category(['name' => 'Processors']);
$category->save();
$category->products()->saveMany([
    new Product(['name' => 'Intel i5']),
    new Product(['name' => 'Intel i7'])
]);

$product = new Product(['name' => 'Intel i3']);
$product->category()->associate($category);
$product->save();
```

Relaciones *—*

- Para las relaciones muchos a muchos es necesaria una tabla intermedia en la base de datos con el nombre `<tabla1>_<tabla2>`
- Por ejemplo, para relacionar las tablas `users` y `roles` se crea la tabla intermedia `role_user` (el orden se determina alfabéticamente)
- La tabla intermedia debe contener las claves ajenas a las otras dos tablas (p.ej. `user_id` y `role_id`)

Relaciones *—*

- Los dos objetos de la relación usan el método `belongsToMany()`

```
class User extends Model {
    public function roles() {
        return $this->belongsToMany('App\Role');
    }
}

class Role extends Model {
    public function users() {
        return $this->belongsToMany('App\User');
    }
}
```

Relaciones *—*

- Para guardar objetos relacionados se usa el método `attach()` en cualquiera de los dos lados de la relación

```php
$user = new User(['name' => 'root']);
$user->save();

$role = new Role(['name' => 'administrator']);
$role->save();

$user->role()->attach($role->id);
// O la instrucción equivalente
$role->user()->attach($user->id);
```

Acceso a objetos relacionados

- Para acceder a los objetos relacionados se usa el nombre del método como si fuera una propiedad (sin paréntesis)

```
User::find(1)->phone->number

Phone::find(1)->user->name
```

- Esta sintaxis permite navegar de unos objetos a otros a través de sus relaciones sin tener que lanzar consultas explícitamente
- Los objetos relacionados se cargan siguiendo la estrategia **Lazy Load**: sólo se recuperan de la base de datos cuando se accede a ellos por primera vez

Acceso a objetos relacionados

- Cuando en el otro extremo de la relación hay múltiples objetos, la propiedad devuelve una colección

```
$products = Category::find(1)->products;
foreach ($products as $product) {
    echo $product->name;
}
```

- Se pueden encadenar consultas llamando al método de la relación con paréntesis

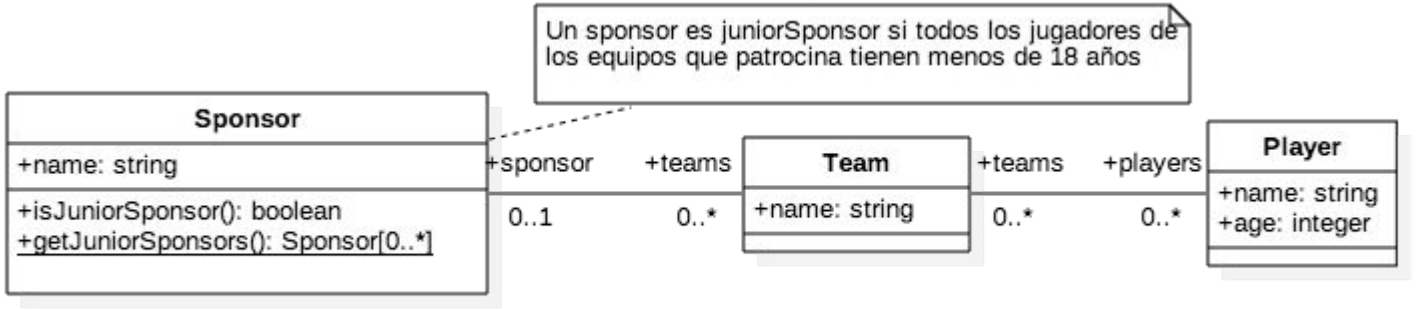
```
$roles = User::find(1)->roles()->orderBy('name')->get();
```

Laravel: Eloquent ORM

EJERCICIO

Ejercicio

Implementa el siguiente modelo en un proyecto Laravel



Recomendaciones:

1. Descarga el proyecto https://github.com/cperezs/dss-eloquent-exercise
2. Crea una migración y un Seeder para cada modelo (es mejor usar Eloquent en lugar de Query Builder en los Seeders), en la siguiente página tienes datos de ejemplo para crear los seeders
3. Crea primero el modelo Sponsor
4. Crea a continuación el modelo Team relacionado con Sponsor
5. Crea el modelo Player
6. Crea la tabla intermedia para la relación entre Team y Player, la clave primaria debe ser el par de claves ajenas a teams y players
7. Implementa las funciones del modelo Sponsor

Ejercicio

Equipo	Sponsor
Screaming Nachos	Samsung
Elemonators	Samsung
E = MC Hammer	Toshiba
Chili Peppers	Toshiba
Low Expectations	Asus
Rescheduled	Intel
Get Your Kicks	AMD

Jugador	Edad	Equipos
John Doe	17	Screaming Nachos E = MC Hammer
Jim Baker	16	Screaming Nachos Elemonators
Nick Alias	21	Chili Peppers
Steve Bacon	17	Low Expectations
Mike Connor	18	Rescheduled
Tim Eater	17	Rescheduled Get Your Kicks
Billy Sheen	17	Low Expectations
Sam Uncle	17	Chili Peppers Elemonators
Rick Allister	19	Get Your Kicks