Prologue
Getting Started
Architecture Concepts
The Basics
Frontend
Security
Digging Deeper
Database
Eloquent ORM
Getting Started
Relationships
Collections
Mutators
API Resources
Serialization
Testing
Official Packages

Eloquent: Serialization

Introduction
Serializing Models & Collections
Serializing To Arrays
Serializing To JSON
Hiding Attributes From JSON
Appending Values To JSON
Date Serialization

Introduction

When building JSON APIs, you will often need to convert your models and relationships to arrays or JSON. Eloquent includes convenient methods for making these conversions, as well as controlling which attributes are included in your serializations.

Serializing Models & Collections

Serializing To Arrays

To convert a model and its loaded relationships to an array, you should use the `toArray` method. This method is recursive, so all attributes and all relations (including the relations of relations) will be converted to arrays:

```
$user = App\User::with('roles')->first();

return $user->toArray();
```

To convert only a model's attributes to an array, use the `attributesToArray` method:

```
$user = App\User::first();

return $user->attributesToArray();
```

You may also convert entire collections of models to arrays:

```
$users = App\User::all();

return $users->toArray();
```

Serializing To JSON

To convert a model to JSON, you should use the `toJson` method. Like `toArray`, the `toJson` method is recursive, so all attributes and relations will be converted to JSON. You may also specify JSON encoding options supported by PHP:

```
$user = App\User::find(1);

return $user->toJson();

return $user->toJson(JSON_PRETTY_PRINT);
```

Alternatively, you may cast a model or collection to a string, which will automatically call the `toJson` method on the model or collection:

```
$user = App\User::find(1);

return (string) $user;
```

Since models and collections are converted to JSON when cast to a string, you can return Eloquent objects directly from your application's routes or controllers:

```
Route::get('users', function () {
    return App\User::all();
});
```

Relationships

When an Eloquent model is converted to JSON, its loaded relationships will automatically be included as attributes on the JSON object. Also, though Eloquent relationship methods are defined using "camel case", a relationship's JSON attribute will be "snake case".

Hiding Attributes From JSON

Sometimes you may wish to limit the attributes, such as passwords, that are included in your model's array or JSON representation. To do so, add a `$hidden` property to your model:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = ['password'];
}
```

When hiding relationships, use the relationship's method name.

Alternatively, you may use the `visible` property to define a white-list of attributes that should be included in your model's array and JSON representation. All other attributes will be hidden when the model is converted to an array or JSON:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that should be visible in arrays.
     *
     * @var array
     */
    protected $visible = ['first_name', 'last_name'];
}
```

Temporarily Modifying Attribute Visibility

If you would like to make some typically hidden attributes visible on a given model instance, you may use the `makeVisible` method. The `makeVisible` method returns the model instance for convenient method chaining:

```php
return $user->makeVisible('attribute')->toArray();
```

Likewise, if you would like to make some typically visible attributes hidden on a given model instance, you may use the `makeHidden` method.

```php
return $user->makeHidden('attribute')->toArray();
```

Appending Values To JSON

Occasionally, when casting models to an array or JSON, you may wish to add attributes that do not have a corresponding column in your database. To do so, first define an accessor for the value:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
```

```
    /**
     * Get the administrator flag for the user.
     *
     * @return bool
     */
    public function getIsAdminAttribute()
    {
        return $this->attributes['admin'] === 'yes';
    }
}
```

After creating the accessor, add the attribute name to the `appends` property on the model. Note that attribute names are typically referenced in "snake case", even though the accessor is defined using "camel case":

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The accessors to append to the model's array form.
     *
     * @var array
     */
    protected $appends = ['is_admin'];
}
```

Once the attribute has been added to the `appends` list, it will be included in both the model's array and JSON representations. Attributes in the `appends` array will also respect the `visible` and `hidden` settings configured on the model.

Appending At Run Time

You may instruct a single model instance to append attributes using the `append` method. Or, you may use the `setAppends` method to override the entire array of appended properties for a given model instance:

```
return $user->append('is_admin')->toArray();

return $user->setAppends(['is_admin'])->toArray();
```

Date Serialization

Customizing The Date Format Per Attribute

You may customize the serialization format of individual Eloquent date attributes by specifying the date format in the cast declaration:

```
protected $casts = [
    'birthday' => 'date:Y-m-d',
    'joined_at' => 'datetime:Y-m-d H:00',
];
```

Global Customization Via Carbon

Laravel extends the Carbon date library in order to provide convenient customization of Carbon's JSON serialization format. To customize how all Carbon dates throughout your application are serialized, use the `Carbon::serializeUsing` method. The `serializeUsing` method accepts a Closure which returns a string representation of the date for JSON serialization:

```
<?php

namespace App\Providers;

use Illuminate\Support\Carbon;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register bindings in the container.
     *
     * @return void
     */
    public function register()
    {
        //
    }
```

```php
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Carbon::serializeUsing(function ($carbon) {
            return $carbon->format('U');
        });
    }
}
```

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.