## Chapitre 4 – Modéliser les relations complexes

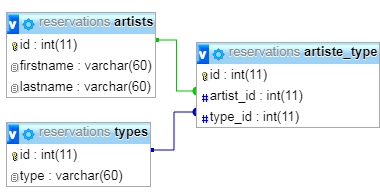
Il est temps d’aborder les relations *ManyToMany* entre tables. C’est le cas respectivement pour la relation *artists* 🡪 *types*, *representations* 🡪 *users, users* 🡪 *roles*, mais aussi *artist\_type* 🡪 *shows*.

<https://laravel.com/docs/7.x/eloquent-relationships#many-to-many>

### Relation *ManyToMany*

Dans ce cas-ci, nous ne créons **ni modèle ni contrôleur** pour cette table ! Commençons par définir la relation ManyToMany dans les modèles *Artist* et *Type*.

1. Modifier les modèles *Artist* et *Type* de façon à ajouter la relation *ManyToMany*.



*app/Artist.php*

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Artist extends Model

{

protected $fillable = ['firstname', 'lastname'];

protected $table = 'artists';

public $timestamps = false;

/\*\*

\* The types that belong to the artist.

\*/

public function types()

{

return $this->**belongsToMany**('App\Type');

}

}

*app/Type.php*

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Type extends Model

{

protected $fillable = ['type'];

protected $table = 'types';

public $timestamps = false;

/\*\*

\* The artists that are defined by the type.

\*/

public function artists()

{

return $this->**belongsToMany**('App\Artist');

}

}

1. Créer et modifier la migration

php artisan make:migration create\_artist\_type\_table

Modifier le fichier généré : ajouter les champs et leurs contraintes. Définir les contraintes de clé étrangère pour le champ *locality\_id*. Ajouter la contrainte d’unicité pour le champ *slug*.

*database\migrations\YYYY\_MM\_DD\_HHMMSS\_create\_artist\_type\_table.php*

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateArtistTypeTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('artist\_type', function (Blueprint $table) {

$table->id();

**$table->foreignId('artist\_id');**

**$table->foreignId('type\_id');**

**$table->foreign('artist\_id')->references('id')->on('artists')**

**->onDelete('restrict')->onUpdate('cascade');**

**$table->foreign('type\_id')->references('id')->on('types')**

**->onDelete('restrict')->onUpdate('cascade');**

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('artist\_type');

}

}

1. Exécuter les migrations

php artisan migrate

1. Créer et modifier une classe de *seeders*

php artisan make:seeder ArtistTypeTableSeeder

*database\seeds\ArtistTypeTableSeeder.php*

<?php

use Illuminate\Database\Seeder;

**use App\ArtistType;**

**use App\Artist;**

**use App\Type;**

class ArtistTypeTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//Empty the table first

DB::statement('SET FOREIGN\_KEY\_CHECKS=0');

**DB::table('artist\_type')->truncate();**

DB::statement('SET FOREIGN\_KEY\_CHECKS=1');

//Define data

$artistTypes = [

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'scénographe',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'scénographe',

],

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Marius',

'artist\_lastname'=>'Von Mayenburg',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Olivier',

'artist\_lastname'=>'Boudon',

'type'=>'scénographe',

],

[

'artist\_firstname'=>'Anne Marie',

'artist\_lastname'=>'Loop',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Pietro',

'artist\_lastname'=>'Varasso',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Laurent',

'artist\_lastname'=>'Caron',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Élena',

'artist\_lastname'=>'Perez',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Guillaume',

'artist\_lastname'=>'Alexandre',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Claude',

'artist\_lastname'=>'Semal',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Laurence',

'artist\_lastname'=>'Warin',

'type'=>'scénographe',

],

[

'artist\_firstname'=>'Claude',

'artist\_lastname'=>'Semal',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Pierre',

'artist\_lastname'=>'Wayburn',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Gwendoline',

'artist\_lastname'=>'Gauthier',

'type'=>'auteur',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'scénographe',

],

[

'artist\_firstname'=>'Pierre',

'artist\_lastname'=>'Wayburn',

'type'=>'comédien',

],

[

'artist\_firstname'=>'Gwendoline',

'artist\_lastname'=>'Gauthier',

'type'=>'comédien',

],

];

//Insert data in the table

foreach ($artistTypes as $data) {

$artist = Artist::where([

['firstname','=',$data['artist\_firstname'] ],

['lastname','=',$data['artist\_lastname'] ]

])->first();

$type = Type::firstWhere('type',$data['type']);

DB::table('artist\_type')->insert([

'artist\_id' => $artist->id,

'type\_id' => $type->id,

]);

}

}

}

Attention, nous devons aussi modifier la classe *LocalitiesTableSeeder* puisqu’il y a une contrainte de clé étrangère *ON DELETE restrict*.

1. Modifier les classes de *seeders ArtistsTableSeeder* et *TypesTableSeeder*

Nous désactivons la vérification des contraintes de clés étrangères avant de vider la table, puis nous la réactivons. Attention à ne faire cela qu’en mode développement, pas en production.

//…

**use App\Artist;**

class ArtistsTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//Empty the table first

**DB::statement('SET FOREIGN\_KEY\_CHECKS=0');**

Artist::truncate();

**DB::statement('SET FOREIGN\_KEY\_CHECKS=1');**

//…

//…

**use App\Type;**

class TypesTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//Empty the table first

**DB::statement('SET FOREIGN\_KEY\_CHECKS=0');**

Type::truncate();

**DB::statement('SET FOREIGN\_KEY\_CHECKS=1');**

//…

1. Exécuter les seeders
   1. Dans le fichier *DatabaseSeeder.php*, il faut d’abord ajouter la classe *ArtistTypeTableSeeder* à la liste des seeders à exécuter :

*database\seeds\DatabaseSeeder.php*

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Seed the application's database.

\*

\* @return void

\*/

public function run()

{

$this->call(**[**

ArtistsTableSeeder::class,

TypesTableSeeder::class,

LocalitiesTableSeeder::class,

RolesTableSeeder::class,

LocationsTableSeeder::class,

ShowsTableSeeder::class,

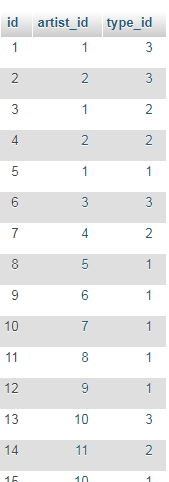
RepresentationsTableSeeder::class,

**ArtistTypeTableSeeder::class,**

**]**);

}

}

* + 1. Ensuite, il faut regénérer le fichier autoloader de Composer !

composer dump-autoload

* + 1. Enfin, nous pouvons exécuter les seeders :

php artisan db:seed

1. Versionner le projet

git status

git add .

git commit -m "Relation ManyToMany between Artist & Type"

### Templates

1. Modifier les templates *show.blade.php*

Dans le dossier *artist*, ajoutons l’affichage des types (fonctions) de l’artiste dans le template *show.blade.php*.

*ressources/views/artist/show.blade.php*

@extends('layouts.app')

@section('title', 'Fiche d\'un artiste')

@section('content')

<h1>{{ $artist->firstname }} {{ $artist->lastname }}</h1>

<h2>Liste des types</h2>

<ul>

**@foreach($artist->types as $type)**

<li>{{ $type->type }}</li>

**@endforeach**

</ul>

@endsection

Dans le dossier *type*, ajoutons l’affichage des artistes qui exercent ce type de fonction dans le template *show.blade.php*.

*ressources/views/type/show.blade.php*

@extends('layouts.app')

@section('title', 'Fiche d\'un type')

@section('content')

<h1>{{ $type->type }}</h1>

<h2>Liste des artistes</h2>

<ul>

**@foreach($type->artists as $artist)**

<li>{{ $artist->firstname }} {{ $artist->lastname }}</li>

**@endforeach**

</ul>

@endsection

1. Lancer le serveur interne (ou accéder directement à l’URL de votre Virtual Host)

php artisan serve

1. Accéder à l’URL <http://localhost:8000/artist/1> (ou <http://laravel.localhost/artist/1>)
2. Accéder à l’URL <http://localhost:8000/type/1> (ou <http://laravel.localhost/type/>1)

1. Versionner le projet

git status

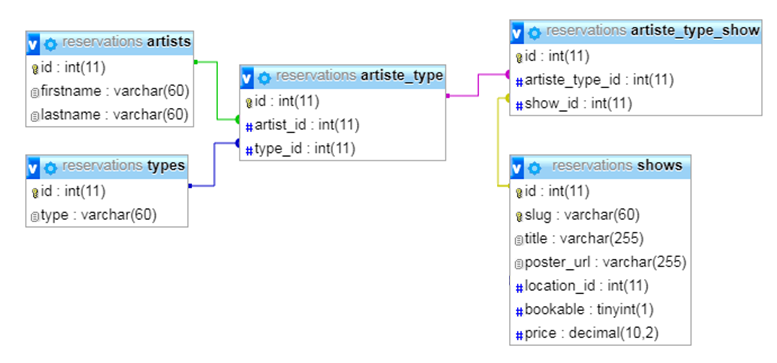
git add .

git commit -m "Update templates Artist & Type"

### Relation *ManyToMany* avec table pivot

Dans ce cas-ci, nous avons finalement besoin de créer le **modèle ArtistType** pour la table pivot *artist\_type*! Ceci afin de pouvoir réaliser une relation *ManyToMany* entre les modèles *ArtistType et Show*.

<https://laravel.com/docs/7.x/eloquent-relationships#many-to-many>



1. Créer le modèle *ArtistType* (et sa relation avec le modèle *Show*)

**php artisan make:model ArtistType**

*app\ArtistType.php*

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class ArtistType extends Model

{

protected $fillable = [

'artist\_id', 'type\_id',

];

protected $table = 'artist\_type';

public $timestamps = false;

/\*\*

\* Get the show of the performance (artist in a type of collaboration for a show)

\*/

**public function shows()**

**{**

**return $this->belongsToMany('App\Show');**

**}**

/\*\*

\* Get the artist for that association.

\*/

public function artist()

{

return $this->**belongsTo**('App\Artist');

}

/\*\*

\* Get the type for that association.

\*/

public function type()

{

return $this->**belongsTo**('App\Type');

}

}

Notez que nous ajoutons aussi les relations inverses *OneToMany* vers les modèles *Artist* et *Type*. Cela nous permettra en partant d’un spectacle *Show* de remonter vers les artistes et les types.

1. Modifier le modèle *Show* de façon à ajouter la relation *ManyToMany*.

*app/Show.php*

//…

/\*\*

\* Get the performance (artist in a type of collaboration) for the show

\*/

**public function artistTypes()**

**{**

**return $this->belongsToMany('App\ArtistType');**

**}**

}

1. Créer et modifier la migration

php artisan make:migration create\_artist\_type\_show\_table

Modifier le fichier généré : ajouter les champs et leurs contraintes. Définir les contraintes de clé étrangère pour le champ *artist\_type\_id* et *show\_id*.

*database\migrations\YYYY\_MM\_DD\_HHMMSS\_create\_artist\_type\_show\_table.php*

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateArtistTypeShowTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('artist\_type\_show', function (Blueprint $table) {

$table->id();

**$table->foreignId('artist\_type\_id');**

**$table->foreignId('show\_id');**

**$table->foreign('artist\_type\_id')->references('id')->on('artist\_type')**

**->onDelete('restrict')->onUpdate('cascade');**

**$table->foreign('show\_id')->references('id')->on('shows')**

**->onDelete('restrict')->onUpdate('cascade');**

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('artist\_type\_show');

} }

1. Exécuter les migrations

php artisan migrate

1. Créer et modifier une classe de *seeders*

php artisan make:seeder ArtistTypeShowTableSeeder

*database\seeds\ArtistTypeShowTableSeeder.php*

<?php

use Illuminate\Database\Seeder;

**use App\Artist;**

**use App\Type;**

**use App\ArtistType;**

**use App\Show;**

class ArtistTypeShowTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//Empty the table first

DB::statement('SET FOREIGN\_KEY\_CHECKS=0');

DB::table('artist\_type\_show')->truncate();

DB::statement('SET FOREIGN\_KEY\_CHECKS=1');

//Define data

$artistTypeShows = [

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'auteur',

'show\_slug'=>'ayiti',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'auteur',

'show\_slug'=>'ayiti',

],

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'scénographe',

'show\_slug'=>'ayiti',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'scénographe',

'show\_slug'=>'ayiti',

],

[

'artist\_firstname'=>'Daniel',

'artist\_lastname'=>'Marcelin',

'type'=>'comédien',

'show\_slug'=>'ayiti',

],

[

'artist\_firstname'=>'Marius',

'artist\_lastname'=>'Von Mayenburg',

'type'=>'auteur',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Olivier',

'artist\_lastname'=>'Boudon',

'type'=>'scénographe',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Anne Marie',

'artist\_lastname'=>'Loop',

'type'=>'comédien',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Pietro',

'artist\_lastname'=>'Varasso',

'type'=>'comédien',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Laurent',

'artist\_lastname'=>'Caron',

'type'=>'comédien',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Élena',

'artist\_lastname'=>'Perez',

'type'=>'comédien',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Guillaume',

'artist\_lastname'=>'Alexandre',

'type'=>'comédien',

'show\_slug'=>'cible-mouvante',

],

[

'artist\_firstname'=>'Claude',

'artist\_lastname'=>'Semal',

'type'=>'auteur',

'show\_slug'=>'ceci-nest-pas-un-chanteur-belge',

],

[

'artist\_firstname'=>'Laurence',

'artist\_lastname'=>'Warin',

'type'=>'scénographe',

'show\_slug'=>'ceci-nest-pas-un-chanteur-belge',

],

[

'artist\_firstname'=>'Claude',

'artist\_lastname'=>'Semal',

'type'=>'comédien',

'show\_slug'=>'ceci-nest-pas-un-chanteur-belge',

],

[

'artist\_firstname'=>'Pierre',

'artist\_lastname'=>'Wayburn',

'type'=>'auteur',

'show\_slug'=>'manneke',

],

[

'artist\_firstname'=>'Gwendoline',

'artist\_lastname'=>'Gauthier',

'type'=>'auteur',

'show\_slug'=>'manneke',

],

[

'artist\_firstname'=>'Philippe',

'artist\_lastname'=>'Laurent',

'type'=>'scénographe',

'show\_slug'=>'manneke',

],

[

'artist\_firstname'=>'Pierre',

'artist\_lastname'=>'Wayburn',

'type'=>'comédien',

'show\_slug'=>'manneke',

],

[

'artist\_firstname'=>'Gwendoline',

'artist\_lastname'=>'Gauthier',

'type'=>'comédien',

'show\_slug'=>'manneke',

],

];

//Insert data in the table

foreach ($artistTypeShows as $data) {

$artist = Artist::where([

['firstname','=',$data['artist\_firstname'] ],

['lastname','=',$data['artist\_lastname'] ]

])->first();

$type = Type::firstWhere('type',$data['type']);

$artistType = ArtistType::where([

['artist\_id','=',$artist->id ],

['type\_id','=',$type->id ]

])->first();

$show = Show::firstWhere('slug',$data['show\_slug']);

DB::table('artist\_type\_show')->insert([

'artist\_type\_id' => $artistType->id,

'show\_id' => $show->id,

]);

}

}

}

Notez que pour trouver l’instance *ArtistType*, nous recherchons d’abord l’artiste par son firstname et son lastname. Puis, nous recherchons le type.

De plus, nous recherchons le spectacle *Show* par son slug.

1. Exécuter les seeders
   1. Dans le fichier *DatabaseSeeder.php*, il faut d’abord ajouter la classe *ArtistTypeShowTableSeeder* à la liste des seeders à exécuter :

*database\seeds\DatabaseSeeder.php*

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Seed the application's database.

\*

\* @return void

\*/

public function run()

{

$this->call(**[**

ArtistsTableSeeder::class,

TypesTableSeeder::class,

LocalitiesTableSeeder::class,

RolesTableSeeder::class,

LocationsTableSeeder::class,

ShowsTableSeeder::class,

RepresentationsTableSeeder::class,

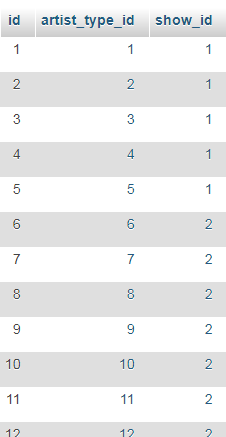
ArtistTypeTableSeeder::class,

**ArtistTypeShowTableSeeder::class,**

**]**);

}

}

* + 1. Ensuite, il faut regénérer le fichier autoloader de Composer !

composer dump-autoload

* + 1. Enfin, nous pouvons exécuter les seeders :

php artisan db:seed

1. Versionner le projet

git status

git add .

git commit -m "Relation ManyToMany between ArtistType & Show"

### Contrôleurs et templates

1. Modifier les templates *show.blade.php*

Dans le dossier *shows*, ajoutons l’affichage des artistes qui collaborent au spectacle (table *artist\_type\_show*) en spécifiant leur fonction (type d’artiste) dans le template *show.blade.php*.

*ressources/views/show/show.blade.php*

@extends('layouts.app')

@section('title', 'Fiche d\'un spectacle')

@section('content')

<article>

<h1>{{ $show->title }}</h1>

//…

<h2>Liste des représentations</h2>

@if($show->representations->count()>=1)

<ul>

@foreach ($show->representations as $representation)

<li>{{ $representation->when }}</li>

@endforeach

</ul>

@else

<p>Aucune représentation</p>

@endif

<h2>Liste des artistes</h2>

<ul>

**@foreach ($show->artistTypes as $collaborateur)**

<li>

**{{ $collaborateur->artist->firstname }}**

**{{ $collaborateur->artist->lastname }}**

**({{ $collaborateur->type->type }})**

</li>

**@endforeach**

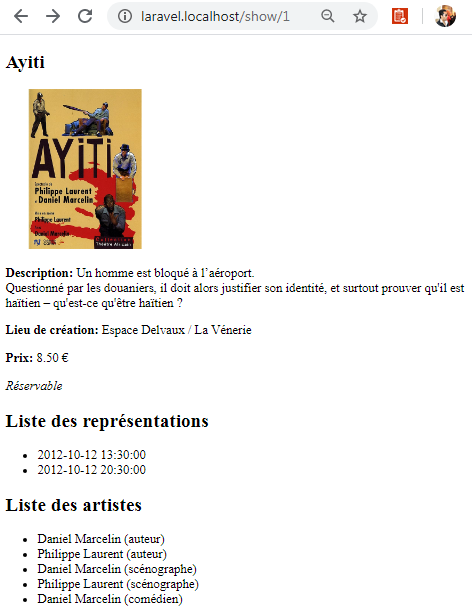
</ul>

@endsection

1. Lancer le serveur interne (ou accéder directement à l’URL de votre Virtual Host)

php artisan serve

1. Accéder à l’URL <http://localhost:8000/show/1> (ou <http://laravel.localhost/show/1>)



Ce résultat n’est pas très satisfaisant. Nous pouvons l’améliorer en modifiant le contrôleur et le template.

1. Modifier le contrôleur *ShowController*

Nous voulons récupérer la liste de toules artistes qui ont collaboré au spectacle en les groupant par type. Ajoutons du code dans la méthode *show*.

*App\Http\Controllers\ShowController.php*

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Show;

class LocalityController extends Controller

{

//…

public function show($id)

{

$show = Show::find($id);

//Récupérer les artistes du spectacle et les grouper par type

$collaborateurs = [];

foreach($show->artistTypes as $at) {

$collaborateurs[$at->type->type][] = $at->artist;

}

return view('show.show',[

'show' => $show,

**'collaborateurs' => $collaborateurs,**

]);

}

//…

}

1. Modifier le template *show.blade.php*

À présent, nous allons pouvoir traiter directement chaque type d’artiste et les afficher proprement dans le template *show.blade.php*.

*ressources/views/show/show.blade.php*

@extends('layouts.app')

@section('title', 'Fiche d\'un spectacle')

@section('content')

<article>

<h1>{{ $show->title }}</h1>

//…

<h2>Liste des représentations</h2>

@if($show->representations->count()>=1)

<ul>

@foreach ($show->representations as $representation)

<li>{{ $representation->when }}</li>

@endforeach

</ul>

@else

<p>Aucune représentation</p>

@endif

<h2>Liste des artistes</h2>

<p><strong>Auteur:</strong>

**@foreach ($collaborateurs['auteur'] as $auteur)**

{{ $auteur->firstname }}

{{ $auteur->lastname }}**@if($loop->iteration == $loop->count-1)** et

**@elseif(!$loop->last)**, **@endif**

**@endforeach**

</p>

<p><strong>Metteur en scène:</strong>

@foreach ($collaborateurs['scénographe'] as $scenographe)

{{ $scenographe->firstname }}

{{ $scenographe->lastname }}@if($loop->iteration == $loop->count-1) et

@elseif(!$loop->last), @endif

@endforeach

</p>

<p><strong>Distribution:</strong>

@foreach ($collaborateurs['comédien'] as $comedien)

{{ $comedien->firstname }}

{{ $comedien->lastname }}@if($loop->iteration == $loop->count-1) et

@elseif(!$loop->last), @endif

@endforeach

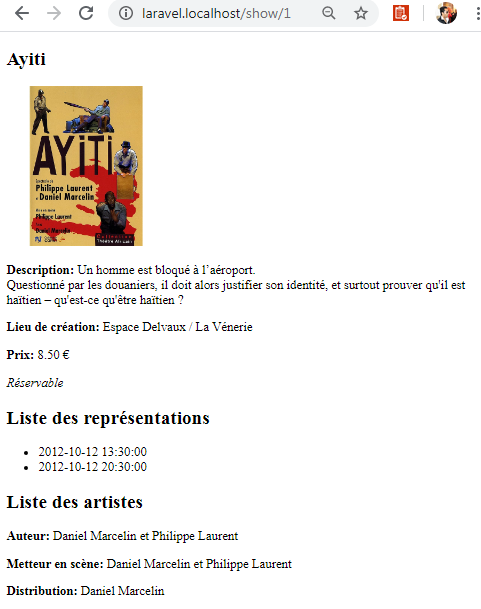
</p>

@endsection

Cette fois, nous contrôlons mieux les artistes que nous voulons afficher. Notez la place précise de la structure *@if-@elseif-@endif* qui nous permet de mettre une virgule ou un « et » entre les artistes.

Si vous trouvez que cela surcharge le template, vous pouvez remonter cette logique dans le contrôleur et envoyer au template une chaîne de caractères formattée avec les virgules et le « et ».

1. Accéder à l’URL <http://localhost:8000/show/1> (ou <http://laravel.localhost/show/1>)
2. Accéder à l’URL <http://localhost:8000/show/2> (ou <http://laravel.localhost/show/2>)



1. Versionner le projet

git status

git add .

git commit -m "Update templates Show (added artists’ list)"

### A vous de jouer !

1. **Répétez ces opérations pour les relations entre les modèles *User* et *Role*.**Notez qu’un utilisateur peut avoir plusieurs rôles. Inversement, un rôle regroupe plusieurs utilisateurs. Il s’agit donc d’une relation ManyToMany.  
   Laravel prévoit déjà une table *users* et son modèle *User*. Dans la migration prédéfinie, ajoutez les champs manquants (*login*, *firstname*, *lastname*, *langue*) et gardez les timestamps ainsi que les champs *email\_verified\_at* et *remember\_token*.  
   Ne définissez pas encore les contrôleurs et les templates (voir chapitre authentification).  
   1. Ajouter les relations ManyToMany entre *User* et *Role*.
   2. Modifier et exécuter la migration
   3. Générer la classe de seeders (données de test)
   4. Définir les données test sans oublier les références dans les classes secondaires
   5. Charger les données test dans la table *role\_user*
2. Versionner le projet
3. **Répétez ces opérations pour les relations entre les modèles *User* et *Representation*.**Un utilisateur peut réserver plusieurs représentations. Inversement, une représentation peut être réservée par plusieurs utilisateurs. Il s’agit aussi d’une relation ManyToMany.
4. Versionner le projet

Bravo!