Atividade Prática 4.1 - Inclusão de Relacionamento N para N Empréstimo

Objetivo
Implementar um relacionamento N para N entre os modelos `User` e `Book` para representar o empréstimo de livros. Utilizaremos o modelo de usuário padrão do Laravel (`App\Models\User`) e criaremos uma tabela intermediária chamada `borrowings` para registrar os detalhes de cada empréstimo.

Instruções

Passo 1: Criar a Migration para a Tabela Intermediária `borrowings`

1.1 Execute o comando para criar a migration da tabela `borrowings`:

```
php artisan make:migration create_borrowings_table
```

1.2 Edite a migration recém-criada em `database/migrations/{timestamp}_create_borrowings_table.php` para definir as colunas necessárias:

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

return new class extends Migration
{
    public function up()
    {
        Schema::create('borrowings', function (Blueprint $table) {
            $table->id();

$table->foreignId('user_id')->constrained()->onDelete('cascade'); //
Relacionamento com User

$table->foreignId('book_id')->constrained()->onDelete('cascade'); //
Relacionamento com Book
            $table->timestamp('borrowed_at')->nullable(); // Data de
Empréstimo
            $table->timestamp('returned_at')->nullable(); // Data de
Devolução
            $table->timestamps();
```

```
        });
    }

    public function down()
    {
        Schema::dropIfExists('borrowings');
    }
};
```

Passo 2: Executar a Migration

2.1 Para criar a tabela no banco de dados, execute o seguinte comando:

```
php artisan migrate
```

Com esses passos, você terá configurado a migration para a tabela `borrowings`, que irá armazenar os registros de empréstimo entre `User` e `Book`. Nos próximos passos, vamos configurar os modelos para estabelecer o relacionamento N para N entre as entidades.

Passo 3: Criar o Model para Empréstimo (Borrowing)

3.1 Execute o comando para criar o model `Borrowing`:

```
php artisan make:model Borrowing
```

3.2 Abra o arquivo `app/Models/Borrowing.php` e configure-o para representar os empréstimos entre `User` e `Book`. Defina os campos que podem ser preenchidos (fillable) e os relacionamentos.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Borrowing extends Model
{
    use HasFactory;
```

```
    // Campos que podem ser preenchidos
    protected $fillable = ['user_id', 'book_id', 'borrowed_at',
'returned_at'];

    // Relacionamento com User
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    // Relacionamento com Book
    public function book()
    {
        return $this->belongsTo(Book::class);
    }
}
```

Esse model `Borrowing` permitirá que você acesse facilmente os dados de `User` e `Book` para cada empréstimo, facilitando a manipulação dos registros de empréstimos. Nos próximos passos, vamos configurar os relacionamentos diretamente nos modelos `User` e `Book` para completar o relacionamento N para N.

Passo 4: Adaptar o Model `User` para o Relacionamento com `Book` através de `Borrowing`

4.1 Abra o arquivo `app/Models/User.php`.

4.2 Adicione o método `books()` para definir o relacionamento N para N entre `User` e `Book` usando a tabela `borrowings` como intermediária:

```
public function books()
{
    return $this->belongsToMany(Book::class, 'borrowings')
                ->withPivot('borrowed_at', 'returned_at')
                ->withTimestamps();
}
```

Esse método permitirá que você acesse os livros emprestados por um usuário usando `$user->books`, além de acessar informações adicionais sobre o empréstimo através do método `withPivot()`.

Passo 5: Adaptar o Model `Book` para o Relacionamento com `User` através de `Borrowing`

5.1 Abra o arquivo `app/Models/Book.php`.

5.2 Adicione o método `users()` para definir o relacionamento N para N entre `Book` e `User` através da tabela `borrowings`:

```php
public function users()
{
    return $this->belongsToMany(User::class, 'borrowings')
                ->withPivot('borrowed_at', 'returned_at')
                ->withTimestamps();
}
```

Esse método permitirá que você acesse os usuários que emprestaram um determinado livro usando `$book->users`, também incluindo informações sobre o empréstimo.

Com essas adaptações, você configurou o relacionamento N para N entre `User` e `Book` através da entidade intermediária `Borrowing`. Agora é possível consultar e manipular os empréstimos entre usuários e livros de forma mais simples no sistema de biblioteca.

Passo 6: Criar Seeds e Factories para `User` e `Borrowing`

6.1 Criar a Factory para Usuários

Não será necessário criar a factory de `User`, pois o Laravel já vem com uma criada por padrão.

6.2 Criar a Factory para `Borrowing`

6.2.1 Execute o comando para criar a factory `BorrowingFactory`:

```
php artisan make:factory BorrowingFactory --model=Borrowing
```

6.2.2 Abra o arquivo `database/factories/BorrowingFactory.php` e configure-o para criar empréstimos entre `User` e `Book` com datas de empréstimo e devolução aleatórias:

```
namespace Database\Factories;

use App\Models\Borrowing;
use App\Models\User;
use App\Models\Book;
use Illuminate\Database\Eloquent\Factories\Factory;

class BorrowingFactory extends Factory
{
    protected $model = Borrowing::class;

    public function definition()
    {
        return [
            'user_id' => User::factory(), // Cria um novo usuário ou usa
um existente
            'book_id' => Book::inRandomOrder()->first()->id, //
Seleciona um livro aleatório
            'borrowed_at' => $this->faker->dateTimeBetween('-1 month',
'now'), // Data de empréstimo
            'returned_at' =>
$this->faker->optional()->dateTimeBetween('now', '+1 month'), // Data de
devolução opcional
        ];
    }
}
```

6.3 Criar Seeder para Popular Usuários e Empréstimos

6.3.1 Crie o seeder `UserBorrowingSeeder` executando:

```
php artisan make:seeder UserBorrowingSeeder
```

6.3.2 Edite o arquivo `database/seeders/UserBorrowingSeeder.php` para gerar usuários e empréstimos. Neste exemplo, criaremos 10 usuários, e cada um poderá ter de 1 a 5 empréstimos.

```php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\User;
use App\Models\Borrowing;
use App\Models\Book;

class UserBorrowingSeeder extends Seeder
{
    public function run()
    {
        // Criar 10 usuários com empréstimos
        User::factory(10)->create()->each(function ($user) {
            Borrowing::factory(rand(1, 5))->create([
                'user_id' => $user->id,
                'book_id' => Book::inRandomOrder()->first()->id,
            ]);
        });
    }
}
```

6.4 Rodar os Seeders

Para popular o banco de dados com usuários e empréstimos, registre o novo seeder `UserBorrowingSeeder` no `DatabaseSeeder.php`.

6.4.1 Abra `database/seeders/DatabaseSeeder.php` e adicione `UserBorrowingSeeder`:

```php
public function run()
{
    $this->call([
        CategorySeeder::class,
        AuthorPublisherBookSeeder::class,
        UserBorrowingSeeder::class, // Novo seeder adicionado aqui
    ]);
}
```

6.4.2 Execute os seeders para popular o banco de dados:

```
php artisan migrate:refresh --seed
```

Com esses passos, você terá usuários e empréstimos gerados aleatoriamente no banco de dados, permitindo testar o relacionamento N para N no sistema de biblioteca.