Lembra do MVC?
Model-View-Controller.

Lembra do CRUD?
Create-Retrieve-Update-Delete

Formulários
Criar formulários no Laravel é igual a criar formulários no PHP clássico.
A diferença está naqueles dois comandos que precisam estar dentro do formulário, lembram?

```
<form method="POST" action="/filmes/delete/1">
    {{ csrf_field() }} ou @csrf
    {{ method_field('DELETE') }}
</form>
```

Request
Lembra dele?
Quando um formulário enviar as informações de volta para rota é o Request quem recebe essas informações para que possa processá-las.

```
public function adicionarFilme(Request $resquest) {

}
```

Validações
A classe request possui um método super bacana para validar ou verificar as informações antes de gravá-las no banco de dados.

```
public function adicionarFilme(Request $resquest) {
   $request->validade([
      'nome' => 'required|unique:filme|max:250'
   ]);
}
```

E para criar, alterar e excluir um registro da sua tabela usando o Eloquent?

Para incluir um registro na sua tabela basta:

```
$usuario = User::create([
   'nome' => $request->input('nome')
]);
$usuario->save();
```

Para atualizar um registro na sua tabela basta:

```
$usuario = Usuario::find(1);
$usuario->name = $request->input('name');
$usuario->save();
```

E para excluir um registro basta:

```
$usuario = Usuario::find(1);
$usuario->delete();
```

1. Relacionamentos

E se precisássemos obter as informações de um registro, em outra tabela, mas que está relacionado com um registro desta tabela?

O Laravel permite configurar no modelo os relacionamentos que ele possui dentro do seu banco de dados dentro do Model.
E a partir dessa configuração ele já consegue também fazer os SELECTs necessários no banco de dados quando tiver que acessar as informações desse outro registro.

Temos que lembrar dos tipos de relacionamentos que existem:

1:1 - um para um - `hasOne()`
1:N - um para muitos - `hasMany()`
N:N - muitos para muitos - `belongsToMany()`

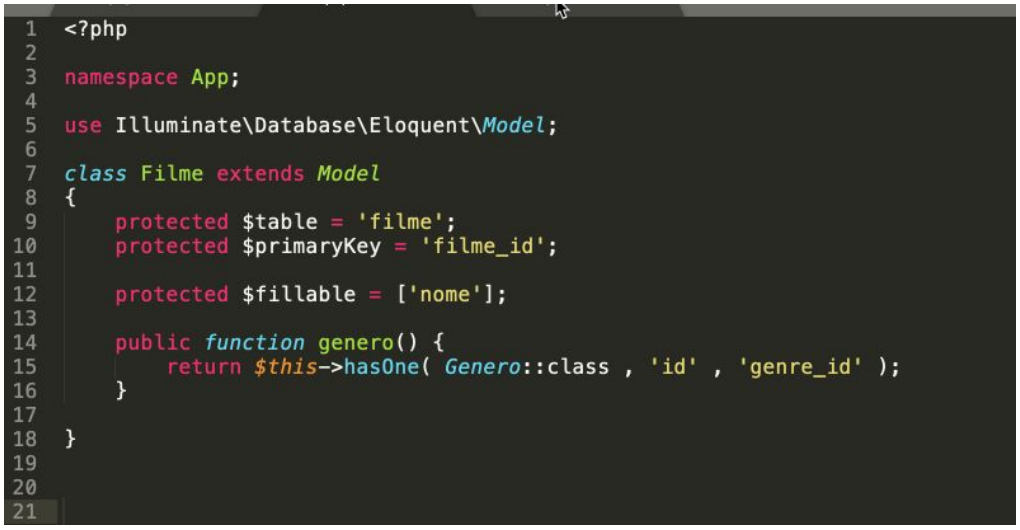
2. 1:1 - 1 para 1

Para indicar que uma determinada tabela do banco de dados possui uma relação de 1 para 1 com outra tabela, devemos ir até o Model que criamos para a primeira tabela e construir uma function usando o `hasOne()`.

Então se quisermos criar uma relação do tipo:

Um filme possui apenas uma categoria relacionada

Vamos ter que fazer assim:



```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Filme extends Model
{
    protected $table = 'filme';
    protected $primaryKey = 'filme_id';

    protected $fillable = ['nome'];

    public function genero() {
        return $this->hasOne( Genero::class , 'id' , 'genre_id' );
    }

}
```

Neste caso criamos uma function chamada "categoria", que vai retornar as informações da categoria relacionada ao filme.

Esse método `hasOne()` possui os seguintes parâmetros:

```
hasOne( <Model que deseja relacionar> , <chave estrangeira> , <chave primária> )
```

3. 1:N - 1 para Muitos

Para indicar que uma determinada tabela do banco de dados possui vários "filhos". Como por exemplo:

Uma categoria possui vários filmes relacionados

Vamos precisar fazer uso do método `hasMany()`.

Fica assim:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Categoria extends Model
{
    protected $table = 'categoria';
    protected $primaryKey = 'categoria';


    public function filmes() {
        return $this->hasMany( Filme::class , 'categoria_id' , 'id' );
    }

}
```

Neste caso criamos uma function chamada "filmes", que quando for utilizada vai retornar todos os filmes relacionadas a categoria.

Esse método possui os seguintes parâmetros:

```
hasMany( <Model que deseja relacionar> , <chave estrangeira> , <chave primária> )
```

4. N:N - Muitos para Muitos

Esta forma de relacionamento é um pouco diferente.
Para esse tipo de relação vamos precisar ter uma terceira tabela.

É o caso, por exemplo, de uma relação desse tipo:

Um filme possui vários artistas, e um artista pertence a vários filmes

Para configurar essa relação vamos precisar do método `belongsToMany()`.

Esse método possui os seguintes parâmetros:

```
belongsToMany( <Model que deseja relacionar> , <tabela intermediaria> , <chave estrangeira> , <chave primária> )
```

Olha como fica:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Filme extends Model
{
    protected $table = 'filme';
    protected $primaryKey = 'filme_id';

    protected $fillable = ['nome'];

    public function atores() {
        return $this->belongsToMany( Ator::class , 'ator_filme' , 'ator_id' , 'filme_id' );
    }
}
```

Percebeu?

Dessa forma o Laravel consegue já criar todas as consultas necessárias no banco de dados automaticamente =)