



Marcos Rodrigo Momo, M.Sc.
e-mail: marcos.momo@ifsc.edu.br

Programação para Internet II



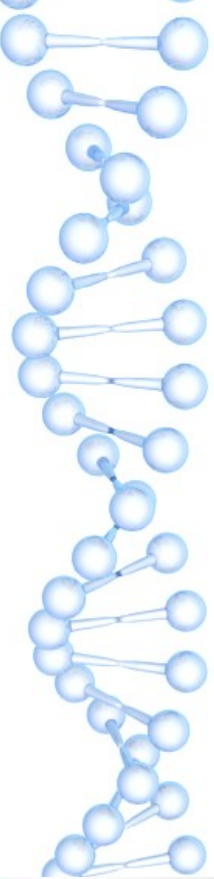
**INSTITUTO
FEDERAL**
Santa Catarina

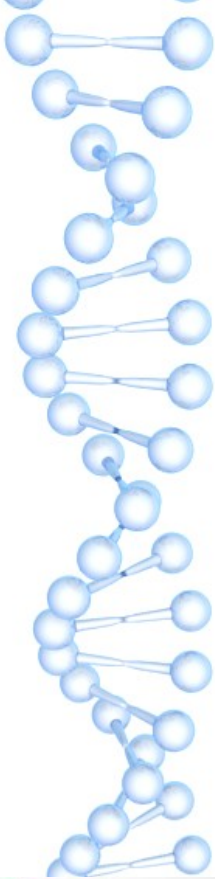
Câmpus
Gaspar

Gaspar, maio 2021.

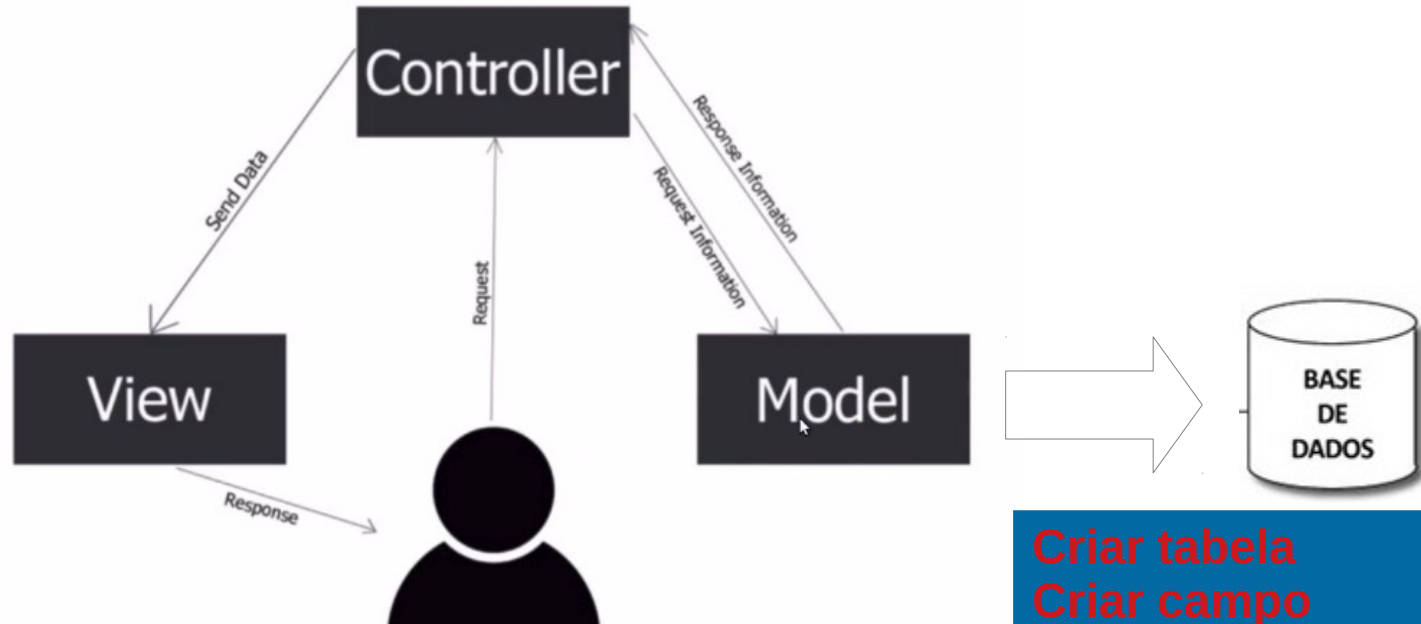
Roteiro

- Banco de dados no Laravel
 - Migrations no Laravel
 - Criando uma base de dados
 - Adicionando tabela
 - Adicionando campos
 - Relacionamentos
 - Comandos para BD
 - Atividades práticas





Model-View-Controller



Atividades

- Criar um banco de dados no mysql chamado aula2205
- Criar um projeto no Laravel
- Vamos criar a tabela produtos
- Vamos adicionar campos a tabela de produtos
- Vamos atualizar a tabela (adicionar mais campos)
- Vamos criar a tabela de vendas
- Vamos adicionar campos a tabela de vendas

Migrations

- Migrations no Laravel permitem a criação e manipulação de bancos de dados
- O objetivo é fornecer uma série de recursos que facilitam o gerenciamento de uma base de dados
- Como migrations podemos manter um histórico de alterações
- Permite reverter qualquer alteração feita
- Auxilia no gerenciamento das alterações realizadas ao banco de dados
- Mantém um controle de controle de versão.

Criando uma base de dados mysql

- Já deve ter um usuário para o mysql, por exemplo:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'user'@'localhost' IDENTIFIED BY 'user';
```

- No windows
 - Podemos utilizar o Xampp e a interface do phpMyAdmin
- No linux, via terminal
 - mysql – u user -p //logar com usuário user
 - Password //informar a senha
 - create database Laravel_products; // criar um base Laravel_products
 - use Laravel_products; //setar base
 - show tables; //mostrar tabelas
 - quit //sair
 -

Introdução migrations

- O Laravel possibilita que toda estrutura do base de dados seja carregado no projeto
- Posso descrever uma tabela inteira, campo, relacionamento ou a criação de vários campos e vários relacionamentos
- Vamos criar um projeto chamado
- Essas configurações ficam no arquivo .env
 - database > migrations



Vamos criar um novo projeto

laravel new vendas

Arquivo .env

- Configurar:
 - Conexão
 - Host
 - Porta
 - Nome da base
 - Usuário
 - Senha

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_produtos
DB_USERNAME=user
DB_PASSWORD=user
```



Vamos criar um migrations

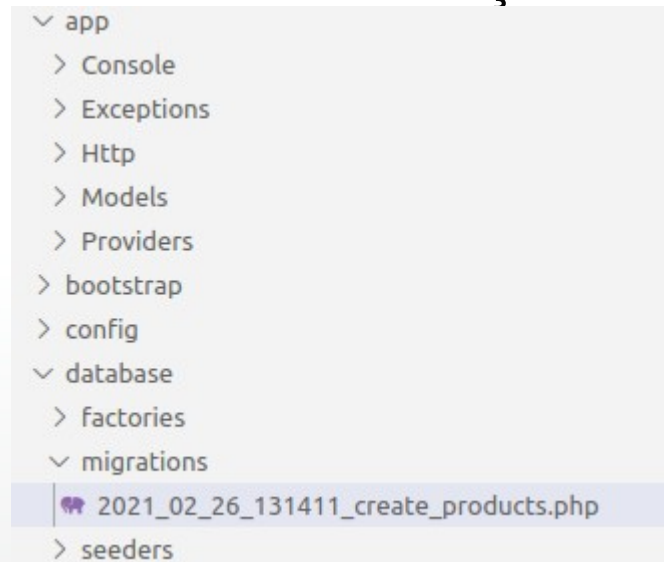
- Através da linha de comando
`php artisan make:migration create_brands`
- Um migration pode descrever:
 - Uma tabela
 - Um campo
 - Um relacionamento
 - Várias tabelas, vários campos, vários relacionamentos

Vamos criar um migrations

- O código utiliza o comando artisan do Laravel para criar uma migration
- Ela nada mais é que uma classe que irá tornar possível a manipulação da tabela de produtos que será criada na base de dados
- Além do comando `php artisan make:migration criar_tabela_products`, ele faz uma chamada para `create=products` que define o nome da tabela a ser criada na base de dados a partir dessa migration.

Migrations

- Uma vez criado um migration, o Laravel cria um arquivo dentro da pasta migrations
- O Laravel, por padrão, traz alguns arquivos de exemplo nesse diretório.
- Observe que ele guarda a data de criação



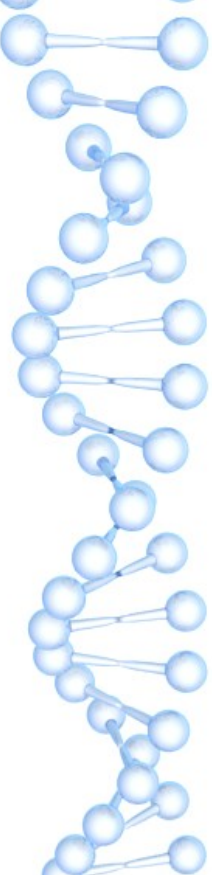
Estrutura do migration

- A classe criada é uma Migration que possui a estrutura básica de métodos e pacotes já definidas pelo Laravel
- Essa classe ficará responsável por criar uma tabela chamada products no banco de dados:

Estrutura do migration

- O código inicia informando quais classes deverão ser utilizadas para construir uma migration, sendo elas:

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



```
use Illuminate\Database\Migrations\Migration;
```

- Migration é uma classe abstrata que é utilizada para definir o tipo da classe que está sendo criada e que deverá estender à classe Migration.



use Illuminate\Database\Schema\Blueprint;

- A classe Blueprint oferece ações que são pertinentes às entidades em si:
 - Por exemplo, a partir dessa classe é possível acessar métodos que irão definir como as colunas dessa tabela deve se comportar,
 - A definição de tipos, tamanhos, e controles de chaves primárias e estrangeiras também são de responsabilidade dessa classe.



use Illuminate\Support\Facades\Schema;

- Fornece um Facade para que seja possível manipular quais ações serão executadas no banco de dados
- No caso, para a função up() esse facade é utilizado para executar a criação de uma tabela no banco de dados
- Na função down() o mesmo é serve também para executar um drop em uma tabela existente no banco.

Função up()

- Esse método será o responsável por criar a tabela de products na base de dados
- A implementação do método prossegue fazendo uso do Facade Schema, demonstrado abaixo:
- O uso do Facade consiste na definição da ação que deseja executar no banco de dados, no caso é desejável criar uma tabela então é utilizado o método create, em seguida é criada uma função anônima que recebe em seu parâmetro através da técnica de injeção de dependência a classe Blueprint sendo referenciada através da variável \$table.
- Com o objeto sendo passado para o contexto da função é possível interagir com seus métodos para informar quais colunas, tipos e respectivos tamanhos deverão ser criados, veja o código e como essa interação acontece:

Função up()

- A primeira linha irá criar um campo chamado ID do tipo inteiro com auto incremento
- A segunda linha, cria colunas para data de criação, data de atualização e data de exclusão que devem armazenar valores do tipo DATETIME.

```
public function up()  
{  
    Schema::create('products', function (Blueprint $table) {  
        $table->bigIncrements('id');  
        $table->timestamps();  
    });  
}
```

Função down()

- O método a classe é public function down() que utiliza o Facade Schema para, caso exista, deletar a tabela que é passada como parâmetro, sendo demonstrado no código abaixo:

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

Inserindo campos na tabela products

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->integer('stock');
        $table->double('price');
        $table->string('description');
        $table->timestamps();
    });
}
```

Executando uma migration

- Com a classe responsável por executar a migration é possível executar o seguinte comando em um terminal:
 - `php artisan migrate`
- O comando acima será responsável por executar as classes que estiverem localizadas no pacote `migration/database/migrations/`
- Caso não seja identificado nenhum erro será criada a tabela `Products` na base de dados que a aplicação estiver conectada
- A imagem abaixo demonstra o banco de dados após a execução da migration:

Executando uma migration

- A imagem abaixo demonstra a base de dados chamada laravel_produtos, criada anteriormente pelo terminal (linux) ou phpMyadmin (windows)

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| laravel_produtos |
| mysql |
| performance_schema |
| sys |
+-----+
```

Executando uma migration

- A imagem abaixo demonstra a base de dados após execução do migration
 - php artisan migrate
 - Ele criará a tabela products e os campos conforme definido na função up()

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->integer('stock');
        $table->double('price');
        $table->string('description');
        $table->timestamps();
    });
}
```

```
mysql> show tables;
+-----+
| Tables_in_laravel_produtos |
+-----+
| products                    |
+-----+
1 row in set (0.00 sec)
```

```
mysql> describe products;
+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id         | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255)        | NO   |     | NULL    |                 |
| stock      | int(11)             | NO   |     | NULL    |                 |
| price      | double              | NO   |     | NULL    |                 |
| description | varchar(255)        | NO   |     | NULL    |                 |
| created_at | timestamp           | YES  |     | NULL    |                 |
| updated_at | timestamp           | YES  |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```


Atualizando um migration

- Equipes que trabalham no desenvolvimento de software utilizando migration tem ganhos diretos no que diz respeito a manutenibilidade do banco de dados
- Por exemplo, supondo que seja desejável inserir uma nova coluna chamada origem, para realizar essa ação basta adicionar a seguinte linha ao código da classe de migration:
-

Atualizando um migration

- Adicionando campo na tabela products

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->integer('stock');
        $table->double('price');
        $table->string('description');
        $table->string('origem', 30);
        $table->timestamps();
    });
}
```

Executando
php artisan migrate:refresh

```
$ php artisan migrate:refresh
Rolling back: 2021_02_26_132051_create_brands
Rolled back: 2021_02_26_132051_create_brands (25.95ms)
Rolling back: 2021_02_26_131411_create_products
Rolled back: 2021_02_26_131411_create_products (25.45ms)
Migrating: 2021_02_26_131411_create_products
Migrated: 2021_02_26_131411_create_products (48.04ms)
Migrating: 2021_02_26_132051_create_brands
Migrated: 2021_02_26_132051_create_brands (46.66ms)
$
```

Resultado do comando php artisan make:refresh

```
mysql> desc products;
```

Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
stock	int	NO		NULL	
price	double	NO		NULL	
description	varchar(255)	NO		NULL	
origem	varchar(30)	NO		NULL	
created_at	timestamp	YES		NULL	
updated_at	timestamp	YES		NULL	

8 rows in set (0,00 sec)



Criando uma nova tabela

- Vamos criar uma nova tabela chamada brands com dois campos id e nome
- Primeiro vamos executar o comando make:migration
 - php make:migration create-brands
 - Se não der erros vai sair a mensagem de criação do arquivo migrations

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
$ php artisan make:migration create_brands  
Created Migration: 2021_02_26_132051_create_brands  
$
```

▼ migrations

```
🐘 2021_02_26_131411_create_products.php  
🐘 2021_02_26_132051_create_brands.php
```

Criando uma nova tabela

- Vamos editar o arquivo create_brands de migration recém criado, adicionando os campos da tabela

```
public function up()
{
    Schema::create('brands', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->timestamps();
    });
}
```

▼ migrations

2021_02_26_131411_create_products.php
2021_02_26_132051_create_brands.php

- Executar o comando para a criação da tabela brands
 - php artisan migrate
 - Verifique se a tabela foi criada

```
mysql> show tables;
+-----+
| Tables_in_laravel_produtos |
+-----+
| brands                       |
| migrations                   |
| products                    |
+-----+
3 rows in set (0.00 sec)
```

Desfazendo um migration

- Com migrations também é possível desfazer uma ação que tenha sido realizada no banco de dados para isso o Laravel fornece o comando:
 - `php artisan migrate:rollback`
- Nesse caso é executado a função `down()` do arquivo `migrate`

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

Executando rollback

Antes

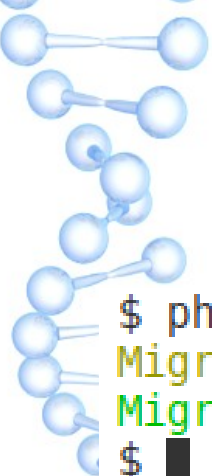
```
mysql> show tables;
+-----+
| Tables_in_produtos |
+-----+
| brands              |
| migrations          |
| products            |
+-----+
3 rows in set (0,01 sec)
```

```
$ php artisan migrate:rollback
Rolling back: 2021_02_26_132051_create_brands
Rolled back: 2021_02_26_132051_create_brands (28.25ms)
$
```

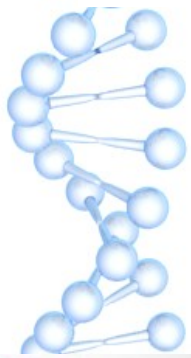
Depois

```
mysql> show tables;
+-----+
| Tables_in_produtos |
+-----+
| migrations          |
| products            |
+-----+
2 rows in set (0,00 sec)
```

Criando novamente a tabela brands



```
$ php artisan migrate
Migrating: 2021_02_26_132051_create_brands
Migrated: 2021_02_26_132051_create_brands (42.75ms)
$
```



```
mysql> show tables;
+-----+
| Tables_in_produtos |
+-----+
| brands              |
| migrations          |
| products            |
+-----+
3 rows in set (0,00 sec)
```


Atividade 1 – não precisa postar

- Baseado na aula de hoje, fazer as seguintes operações na base de dados:
 - Incluir um campo chamado descrição na tabela brands, utilizando o comando de migrate:refresh
 - Fazer alterações e desfazer, utilizando o comando migrate:rollback

Atividade 2 – não precisa postar.....

- Criar uma base de dados chamada alunos
 - Tabela aluno
 - Campos: nome, rua, bairro, cidade, estado, telefone, e-mail, cpf
 - Tabela disciplina
 - Campos: nome, media_obtida
 - Tabela curso
 - Campos: nome, descrição, tempo_duração