

**CURSOS  
TÉCNICOS**

**DESENVOLVIMENTO DE  
SISTEMAS WEB II**

**Eixo Informática para Internet**

**Unidade 7**

## SUMÁRIO

### UNIDADE 7

|  |    |
|--|----|
| 1. Arquitetura MVC e Eloquent ORM.....                           | 3  |
| 1.1. Recapitulando a última unidade.....                         | 3  |
| 1.1.1 Diretório Database do projeto Laravel.....                 | 3  |
| 1.2 Migrations.....  | 3  |
| 1.2.1 Criando migrations.....                                    | 4  |
| 1.2.2 Estrutura das migrations.....                              | 4  |
| 1.2.3 Criando tabelas.....                                       | 5  |
| 1.2.4 Efetuando uma migração.....                                | 5  |
| 1.2.5 Revertendo migrações.....                                  | 6  |
| 2. Seeders.....  | 7  |
| 2.1 Factories.....   | 7  |
| 2.2 Models.....  | 8  |
| 3. Eloquent ORM.....   | 8  |
| 3.1 Criando classes de Modelo.....                               | 8  |
| 4. Convenções do Eloquent Models.....                            | 9  |
| 4.1 Nome de tabelas.....   | 9  |
| 4.2 Primary key.....   | 9  |
| 4.3 Timestamps.....  | 10 |
| 4.4 Valores padrões de atributos.....                            | 10 |
| 5. Recuperando todos registros de um model .....                 | 11 |
| 5.1 Recuperando registros de um model com query estruturada..... | 15 |
| 5.2 Inserindo um novo registro com Eloquent Model.....           | 17 |
| 6. Fixando o conhecimento: Desafio.....                          | 19 |
| 7. Referências.....  | 20 |

## UNIDADE 7

### 1. Arquitetura MVC e Eloquent ORM

Nesta unidade, vamos focar na arquitetura MVC e Eloquent ORM, e iniciaremos por um retrospecto da unidade anterior.

#### 1.1. Recapitulando a última unidade

Na unidade 6, foi apresentado tanto a arquitetura de um projeto MVC quanto um framework moderno para projetos PHP. Vimos, especificamente, sobre Controllers, Views e rotas, porém ainda não realizamos a integração de fato com nosso projeto Laravel e um banco de dados.

Nesta semana, vamos explorar mais do MVC focando na camada Model, e também, vamos aprender algumas outras funcionalidades do Laravel, que podem agregar ainda mais nas aplicações.

##### 1.1.1 Diretório Database do projeto Laravel

O diretório **database** contém suas **database migrations** (migração da base de dados), **model factories** (criadores de modelos) e **seeds**. Vamos explorar mais sobre tudo isso.

#### 1.2 Migrations

Segundo a documentação oficial do Laravel, “**As migrações são como controle de versão para o seu banco de dados**, permitindo que sua equipe defina e compartilhe a definição do esquema do banco de dados da aplicação. Se você já teve que pedir a um colega de equipe para adicionar manualmente uma coluna ao esquema do banco de dados local deles após puxar suas alterações do controle de origem, você enfrentou o problema que as migrações de banco de dados resolvem.”

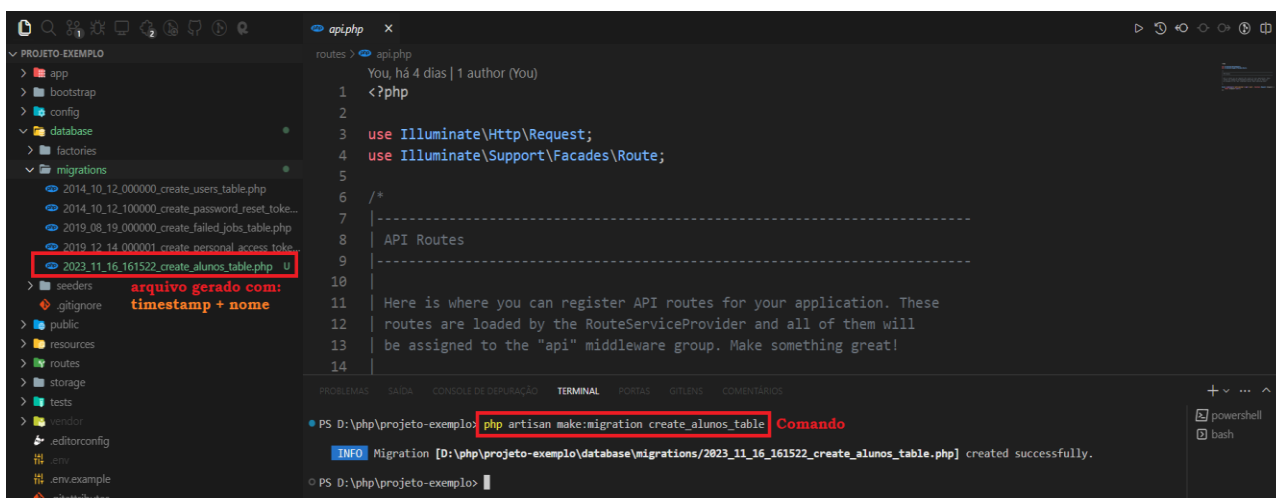
A **Schema facades** do Laravel oferece suporte independente de banco de dados para criar e manipular tabelas em todos os sistemas de banco de dados suportados pelo Laravel. Normalmente, as migrações usam essa fachada para criar e modificar tabelas e colunas no banco de dados.

**Saiba mais sobre facades:** <https://laravel.com/docs/10.x/facades>

### 1.2.1 Criando migrations

Para criar uma **migration**, você deve utilizar o **comando do artisan** **make:migration**, ou seja, `php artisan make:migration nome_migration`. Com isso, será gerado um novo arquivo de **migration** em **database/migration**. Cada arquivo de migration contém um nome com um timestamp gerado na criação, o que permite o Laravel determinar qual a ordem das migrações, possibilitando desfazer uma migração, por exemplo, similar ao que ocorre com o histórico de commits do git.

**Exemplo** - Criação de migration para criar a tabela de alunos

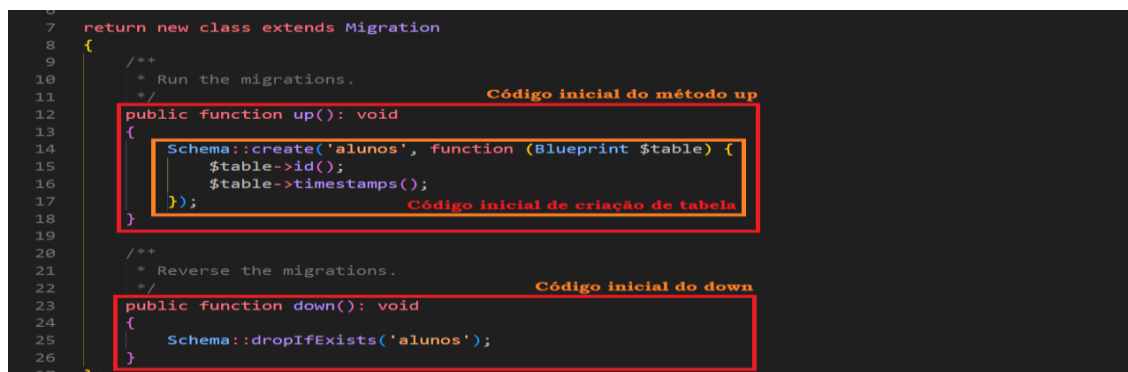


Utilizamos o comando: **php artisan make:migration create\_alunos\_table**. Note que, foi gerado um arquivo com o **timestamp + nome indicado no comando**.

**Ex:** 2023\_11\_16\_161522\_create\_alunos\_table.php

### 1.2.2 Estrutura das migrations

No Laravel, uma classe de migração contém dois métodos: **up** e **down**. O método up é utilizado para adicionar novas tabelas, colunas ou índices ao seu banco de dados, enquanto o método down deve reverter as operações realizadas pelo método up. Confira a estrutura:



### 1.2.3 Criando tabelas

Com os métodos **up** e **down** você pode utilizar o construtor de esquemas do Laravel para criar e modificar tabelas de forma expressiva. Para conhecer todos os métodos disponíveis no **Schema builder**, [consulte a documentação oficial](#).

**Exemplo** - Criação de tabela de alunos usando o **Schema builder**.

| tb_alunos   |                                   |
|-------------|-----------------------------------|
| cd_aluno    | integer not null PK autoincrement |
| nm_aluno    | varchar(255) not null             |
| nm_curso    | varchar(255) not null             |
| nu_ano      | integer not null                  |
| nu_semestre | integer DEFAULT 1                 |
| created_at  | timestamp                         |
| updated_at  | timestamp                         |

Vamos utilizar o exemplo da tabela de alunos construído no desafio da semana 5.

Os campos **created\_at** e **updated\_at** são utilizados, a fim de gerar um histórico.

Estes campos já são fornecidos, por padrão, utilizando timestamp().

**Exemplo** - Comandos de criação da tabela “tb\_alunos”.

```

public function up(): void
{
    comando criação nome tabela identificador
    Schema::create('tb_alunos', function (Blueprint $table) {
        $table->id('cd_aluno')->autoIncrement();
        $table->string('nm_aluno', 255);
        $table->string('nm_curso', 255);
        $table->integer('nu_ano');
        $table->integer('nu_semestre')->default(1)->nullable();
        $table->timestamps();
    });
}
  
```

**colunas com suas definições**

Note que utilizamos o **Schema::create** definimos um **nome da tabela**, **identificador** e através do identificador definimos as colunas com os comandos pré-definidos de criação de tabelas com o Laravel. É possível consultar os tipos de colunas disponíveis no **Schema builder**: <https://laravel.com/docs/10.x/migrations#available-column-types>.

### 1.2.4 Efetuando uma migração

Para de fato migrar todos os arquivos de **migrations**, é necessário executar o comando do Artisan **migrate**. Assim, o Laravel identifica quais arquivos foram modificados e migra as mudanças com o banco de dados. **Se não houver nenhuma mudança para ser migrada no banco, o Laravel emitirá uma mensagem dizendo que não há migrações para ocorrer.**

**Exemplo** - Executando o comando **php artisan migrate**.

```

12 public function up(): void
13 {
14     Schema::create('tb_alunos', function (Blueprint $table) {
15         $table->id('cd_aluno')->autoIncrement();
16         $table->string('nm_aluno', 255);
17         $table->string('nm_curso', 255);
18         $table->integer('nu_ano');
19         $table->integer('nu_semestre')->default(1)->nullable();
20         $table->timestamps();
21     });
22 }
    
```

```

PS D:\php\projeto-exemplo> php artisan migrate
INFO Running migrations.
2023_11_16_161522_create_alunos_table ..... 52ms DONE
    
```

**Exemplo** - Conferir o status das migrações com comando **php artisan migrate:status**

```

PS D:\php\projeto-exemplo> php artisan migrate:status
    
```

| Migration name  | Batch | Status |
|---|-------|--------|
| 2014_10_12_000000_create_users_table                  | [1]   | Ran    |
| 2014_10_12_100000_create_password_reset_tokens_table  | [1]   | Ran    |
| 2019_08_19_000000_create_failed_jobs_table            | [1]   | Ran    |
| 2019_12_14_000001_create_personal_access_tokens_table | [1]   | Ran    |
| 2023_11_16_161522_create_alunos_table                 | [2]   | Ran    |

Como é possível ver na imagem, todos os arquivos e os devidos status são evidenciados.

### 1.2.5 Revertendo migrações

Para reverter a última operação de migração, você pode usar o comando rollback do Artisan. Este comando desfaz a última "batch" de migrações, que pode incluir vários arquivos de migração.

Isso irá desfazer os comandos, como criação de tabelas da última migração.

```

PS D:\php\projeto-exemplo> php artisan migrate:rollback
INFO Rolling back migrations.
2023_11_16_161522_create_alunos_table ..... 6ms DONE
    
```

Para conferir mais comandos específicos de rollbacks das migrations acesse:

→ <https://laravel.com/docs/10.x/migrations#rolling-back-migrations>

**NOTA:** É importante destacar que existem **muitos comandos disponíveis** para a criação, modificação, alteração na estrutura de tabelas, migrações e manipulações de migrações, o que depende muito do contexto da sua aplicação para a utilização ou não destes comandos, então mais uma vez:

**É de extrema importância consultar a documentação oficial do framework para mais detalhes.**

→ <https://laravel.com/docs/10.x/migrations#>

## 2. Seeders

O Laravel inclui a capacidade de popular seu banco de dados com dados usando classes de seed. Todas as classes de seed são armazenadas no diretório database/seeders. Por padrão, a classe **DatabaseSeeder** é definida para você. A partir desta classe, você pode usar o método call para executar outras classes de seed, permitindo que você controle a ordem de semeadura.

**Exemplo da [doc. Oficial](#)** - Criação de seeder para tabela 'users'.

```

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeders.
     */
    public function run(): void
    {
        DB::table('users')->insert([
            'name' => Str::random(10),
            'email' => Str::random(10).'@gmail.com',
            'password' => Hash::make('password'),
        ]);
    }
}

```

### 2.1 Factories

Especificar manualmente os atributos para cada seeder de modelo pode ser trabalhoso. Em vez disso, você pode usar **factories de modelos** para **gerar de forma conveniente grandes quantidades de registros de banco de dados**. Primeiro, revise a documentação das fábricas de modelos para aprender como definir suas fábricas.

**Exemplo da [doc. Oficial](#)** - Criando 50 usuários em massa no banco de dados

```

use App\Models\User;

/**
 * Run the database seeders.
 */
public function run(): void
{
    User::factory()
        ->count(50)
        ->hasPosts(1)
        ->create();
}

```

## 2.2 Models

No Laravel, os "**models**" desempenham um papel fundamental no acesso e **manipulação dos dados no banco de dados**. Um modelo é uma representação orientada a objetos de uma tabela no banco de dados. Cada modelo está associado a uma tabela específica e fornece uma maneira elegante de realizar operações com essa tabela.

## 3. Eloquent ORM

O Laravel inclui o **Eloquent**, um **mapeador objeto-relacional (ORM)** que **torna a interação com o banco de dados mais agradável**. Ao usar o Eloquent, **cada tabela do banco de dados tem um "Modelo" correspondente** que é utilizado para interagir com essa tabela. Além de **recuperar registros** da tabela do banco de dados, os modelos do Eloquent **permitem inserir, atualizar e excluir** registros da tabela também.

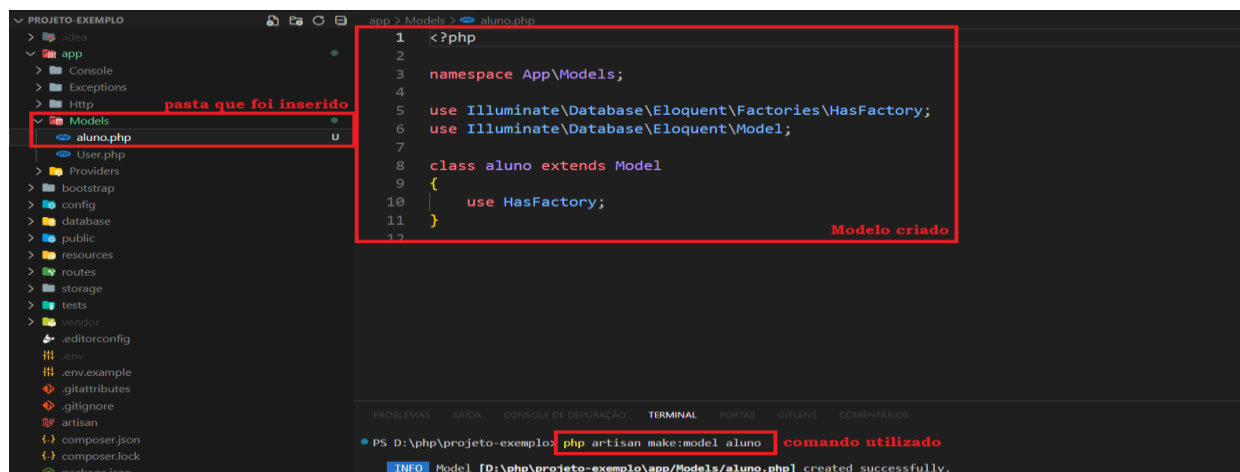
**Lembrete:** Antes de criar seus modelos, lembre-se de **realizar a configuração necessária, para conectar a aplicação Laravel a seu banco de dados**.

Caso ainda não tenha feito isso, confira o material da semana passada OU acesse a [documentação de configuração de banco de dados](#).

### 3.1 Criando classes de Modelo

Para começar você deve criar um **Eloquent model**. Os modelos são armazenados na pasta **app/models** e estendem a classe de modelos do **Eloquent ORM**, que vem de **Illuminate\Database\Eloquent\Model**. Você pode usar o comando do artisan **make:model** para gerar um novo modelo.

**Exemplo** - Criação de modelo "aluno" para a tb\_alunos.



```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class aluno extends Model
9  {
10     use HasFactory;
11 }
    
```

Modelo criado

comando utilizado: `php artisan make:model aluno`

Model [D:\php\projeto-exemplo\app\Models\aluno.php] created successfully.

Com isso, temos uma estrutura inicial para definirmos o nosso modelo.



## 4. Convenções do Eloquent Models

Modelos gerados em Laravel possuem algumas convenções chaves que precisamos entender melhor.

### 4.1 Nome de tabelas

Por padrão, um modelo é vinculado por seu formato de nome em **snake\_case** e no plural.

**Exemplo** - Nomes de modelos e o nome da tabela que será vinculada por padrão.

|                  | Nome modelo   | Nome Tabela    | Explicação   |
|------------------|---------------|----------------|--|
| <b>Exemplo 1</b> | <b>aluno</b>  | <b>alunos</b>  | A tabela vinculada será alunos pois é a forma plural de aluno.   |
| <b>Exemplo 2</b> | <b>nu_ano</b> | <b>nu_anos</b> | A tabela vinculada será desta forma, pois nu_anos é a forma no plural e separado por snake case de nu_ano. |

No entanto, caso o nome da tabela não siga esta convenção **podemos manualmente declarar qual tabela está vinculada a determinado modelo**.

**Exemplo** - Definindo o **model** aluno para estar vinculado na **tabela** "tb\_alunos".

```

app > Models > aluno.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class aluno extends Model
8  {
9
10
11     /**
12      * The table associated with the model.
13      * @var string
14      */
15     protected $table = 'tb_alunos';
16 }
  
```

### 4.2 Primary key

O Eloquent assumirá que cada modelo correspondente nas tabelas do banco de dados terá uma coluna nomeada **"id"**. Se necessário, é possível definir: **protected \$primaryKey = 'nome\_primary\_key'**. Inclusive este atributo denominado \$primaryKey serve para especificar uma chave primária personalizada.

**Exemplo** - Adicionando manualmente o nome da primary key personalizada.

```
5 use Illuminate\Database\Eloquent\Model;
6
7 class aluno extends Model
8 {
9
10     protected $table = 'tb_alunos';
11     protected $primaryKey = 'cd_aluno';
12 }
```

### 4.3 Timestamps

O Eloquent espera que as colunas **created\_at** e **updated\_at** existam no modelo correspondente à base de dados. Isso faz com que o Eloquent, automaticamente, defina os valores destas colunas quando os são **criados OU atualizados**. Caso você não queira que essas colunas sejam automaticamente manipuladas pelo Eloquent, você deve definir **\$timestamps = false;** como atributo do seu modelo.

**Exemplo** - Adicionando na **linha 12** atributos que remove a manipulação automática das colunas **created\_at** e **updated\_at** por parte do Eloquent.

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class aluno extends Model
8 {
9
10     protected $table = 'tb_alunos';
11     protected $primaryKey = 'cd_aluno';
12     protected $timestamps = false;
13 }
```

### 4.4 Valores padrões de atributos

Por padrão, uma instância recém-criada de um modelo não conterá nenhum valor de atributo. Se você desejar definir os valores padrão para alguns dos atributos do seu modelo, você pode definir uma propriedade **\$attributes** no seu modelo. Os valores dos atributos colocados no **array \$attributes** devem estar no formato "bruto" ou "armazenável", como se tivessem sido acabados de ser lidos do banco de dados.

**Exemplo** - Iniciando saldo com 0 por padrão, conforme exemplo nas semanas passadas.

```

14     protected $attributes = [
15         'saldo' => 0
16     ];
  
```

**NOTA:** Existem outras convenções que você queira dar uma conferida na documentação oficial.

→ <https://laravel.com/docs/10.x/eloquent#eloquent-model-conventions>

## 5. Recuperando todos registros de um model

Depois de criar um modelo e sua tabela de banco de dados associada, você está pronto para começar a recuperar dados do seu banco de dados. Você pode pensar em cada modelo Eloquent como um construtor de consultas poderoso que permite consultar fluentemente a tabela de banco de dados associada ao modelo. O método **all** do modelo recuperará todos os registros da tabela de banco de dados associada ao modelo:

**Exemplo** - Recuperando registros de aluno na “tb\_alunos”.

1. Configuração do model de aluno:

```

app > Models > aluno.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class aluno extends Model
8  {
9
10     protected $table = 'tb_alunos';
11     protected $primaryKey = 'cd_aluno';
12 }
  
```

2. Vamos alterar o Controller “AlunoController” para somente debugar os valores do resultado da chamada de todos os registros no model Aluno.

```

app > Http > Controllers > AlunosController.php
You, há 2 minutos | 1 author (You)
1  <?php
2  namespace App\Http\Controllers;
3  use Illuminate\Http\Request;
4  use App\Models\Aluno; Importação do modelo
5
6  class AlunosController extends Controller
7  {
8
9      public function index()
10     {
11         dd(Aluno::all()); debug de todos registros do model aluno com "all"
12
13         return view('alunos.index', [
14             'alunos' => $alunos
15         ]);
16     }
17 }
  
```

3. O resultado obtido na rota **localhost/alunos** da aplicação WEB iniciada via **php artisan serve** é:

```
Illuminate\Database\Eloquent\Collection {#301 ▼ // app\Http\Controllers\AlunosController.php:10
  #items: []
  #escapeWhenCastingToString: false
}
```

Um array de itens vazios, afinal de contas não temos nenhum registro cadastrado na “tb\_alunos”.

**Comando dd:** O comando **dd** é uma abreviação para “**Dump and Die**”. Ele é usado para imprimir (dump) informações sobre uma variável ou expressão e encerrar a execução do script imediatamente. O objetivo principal do dd é **depurar** e **inspecionar valores** durante o desenvolvimento. Por isso que o utilizamos no passo 2.

4. Populando dados na tb\_alunos (**vamos utilizar seeders e factories**).

4.1 - Defina a propriedade **HasFactory** no **model**.

```
app > Models > aluno.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class aluno extends Model
9  {
10     use HasFactory;
11
12     protected $table = 'tb_alunos';
13     protected $primaryKey = 'cd_aluno';
14 }
```

4.2 - Crie um factory via comando do artisan **make:factory**.

```
Pichau@rafael MINGW64 /d/php/projeto-exemplo (migrations)
$ php artisan make:factory AlunoFactory --model=Aluno modelo correspondente
INFO Factory [D:\php\projeto-exemplo\database\factories\AlunoFactory.php] created successfully.
```

4.3 - Defina na factory criada, os atributos de forma a serem gerados aleatoriamente, porém seguindo os padrões definidos na tabela. No Laravel há uma gama de comandos para se utilizar para gerar dados fake.

**Exemplo - Atributos para geração de massa de dados fake**

```

database > factories > AlunoFactory.php
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6
7  /**
8   * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Aluno>
9   */
10 class AlunoFactory extends Factory
11 {
12     public function definition(): array
13     {
14         return [
15             'nm_aluno' => fake()->name(),
16             'nm_curso' => fake()->unique()->name(),
17             'nu_ano' => fake()->numberBetween(2000, 2023),
18             'nu_semestre' => fake()->numberBetween(1, 10)
19         ];
20     }
21 }

```

**NOTA:** Observe que utilizamos os métodos pré-definidos do Laravel de acordo com os campos. Seguimos as particularidades de o **nome do curso ser único**, colocamos um **intervalo para o ano do curso** e **número de semestres**. Caso não definirmos estas **constraints** pode dar erro na inserção por conta das regras da tabela.

**4.4 - Crie um seeder para utilizar o factory e gerar os dados**

```

Pichau@rafael MINGW64 /d/php/projeto-exemplo (migrations)
$ php artisan make:seeder AlunoSeeder

INFO Seeder [D:\php\projeto-exemplo\database\seeders\AlunoSeeder.php] created successfully.

```

**4.5 - No arquivo de seeder criado, utilize o modelo e factory para gerar um número pré-definido de registros.**

```

9  class AlunoSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      */
14     public function run(): void
15     {
16         Aluno::factory()->count(50)->create();
17     }
18 }

```

#### 4.6 - Agora podemos voltar a aplicação WEB e observar o retorno do comando

```
Illuminate\Database\Eloquent\Collection {#352 // app\Http\Controllers\AlunosController.php:10
  #items: array:50 [▼
    0 => App\Models\aluno {#354 ▼
      #connection: "mysql"
      #table: "tb_alunos"
      #primaryKey: "cd_aluno"
      #keyType: "int"
      +incrementing: true
      #with: []
      #withCount: []
      +preventLazyLoading: false
      #perPage: 15
      +exists: true
      +wasRecentlyCreated: false
      #escapeWhenCastingToString: false
      #attributes: array:7 [▶]
      #original: array:7 [▶]
      #changes: []
      #casts: []
      #classCastCache: []
      #attributeCastCache: []
      #dateFormat: null
      #appends: []
      #dispatchesEvents: []
      #observables: []
      #relations: []
      #touches: []
      +timestamps: true
      +usesUniqueIds: false
      #hidden: []
      #visible: []
      #fillable: []
      #guarded: array:1 [▶]
    ]
  }
  1 => App\Models\aluno {#355 ▶}
  2 => App\Models\aluno {#356 ▶}
  3 => App\Models\aluno {#357 ▶}
```

dd.

Note que agora **items** contém 50 dados do tipo aluno. Estes dados possuem um formato chamado Collection do Eloquent.

Uma "collection" refere-se a uma instância da classe `Illuminate\Database\Eloquent\Collection`. **Collections são objetos poderosos e flexíveis fornecidos pelo Laravel** para manipular conjuntos de registros retornados de consultas de banco de dados.

**NOTA:** Estas coleções do Eloquent trazem inúmeros dados, além dos valores atribuídos às colunas que podem ser úteis para mostrar em tela ou desenvolver alguma lógica específica.

**4.7 -** No Controller “AlunosController”, agora podemos fazer a listagem de todos os Alunos em tela. Passando esta informação para a View.

```
6 class AlunosController extends Controller
7 {
8     public function index()
9     {
10
11         $alunos = Aluno::all();
12
13         return view('alunos.index', [
14             'alunos' => $alunos
15         ]);
16     }
17 }
```

**4.8 -** Basta na View, apontar para o nome dos alunos.

```

resources > views > alunos > index.blade.php
You, há 3 minutos | 1 author (You)
1 <x-layout title="Alunos">
2   <ul class="list-group">
3     @foreach($alunos as $aluno)
4       <li class="list-group-item"> {{ $aluno->nm_aluno }} </li>
5     @endforeach
6   </ul>
7
8   <a href="series/create" class="btn btn-dark mt-2">Adicionar Aluno</a>
9 </x-layout>
    
```

**4.9** - O resultado em tela é a listagem de toda massa de dados gerada.

| Alunos                   |
|--------------------------|
| Prof. Reagan Gulgowski   |
| Ernestine Will           |
| Jake Gaylord             |
| Aron Renner              |
| Karen Bruen              |
| Irwin Kertzmann          |
| Ulises Emser             |
| Mr. Newton Cormier       |
| Mr. Ford Champlin        |
| Jacinthe Donnelly        |
| Heath Renner             |
| Prof. Cornelius Donnelly |
| Mr. Norbert Hand IV      |
| Prof. Meda Willms        |
| Miss Jermaine Koelpin V  |
| Hailie Jones             |
| Dr. Darby Mante          |
| Miss Valentina Simonis V |
| Margarett Weimann DVM    |
| Prof. Lila Fahey DDS     |
| Noemy Sawayn             |

### **5.1 Recuperando registros de um model com query estruturada**

Caso seja preciso fazer uma busca mais precisa podemos montar uma query (consulta) para o banco de dados especificando os parâmetros que desejamos.

**Exemplo doc. Oficial - Realizando uma consulta com Where(), orderBy() e limit.**

```

$flights = Flight::where('active', 1)
    ->orderBy('name')
    ->take(10)
    ->get();
    
```

**Exemplo** - Limitando a busca de registros o query builder usando take(n) e sortBy.

```
class AlunosController extends Controller
{
    public function index()
    {
        //Com limite de busca
        $alunos = Aluno::all()->sortBy('nm_aluno')->take(10);

        //Com Where
        //$alunos = Aluno::where('nm_aluno', 'Aron Renner')->get();

        return view('alunos.index', [
            'alunos' => $alunos
        ]);
    }
}
```

**Lembre-se:**

**orderBy()** → Utilizado em uma instância do Eloquent logo após Where ou Get.

**sortBy()** → Utilizado para ordenar coleções, NÃO consultas do banco diretamente.

**Efeito produzido em tela:**



Caso você queira baixar o projeto no estado atual acesse o repositório do github abaixo e faça um ZIP dos arquivos clicando no botão “code” OU faça um clone do repositório e entre na branch “migrations”.

→ Repo: <https://github.com/RafaelR4mos/php-laravel-projeto-alunos-exemplo/tree/migrations>



## 5.2 *Inserindo um novo registro com Eloquent Model*

Para a inserção de registros em uma base de dados, também podemos utilizar o Eloquent para facilitar as coisas.

**Exemplo** - Inserindo um aluno na base de dados

Vamos precisar de uma **página HTML**, **definir rotas** e uma **função no Controller** para inserir um novo registro.

1. Página HTML com campos necessários:

→ **Código**

```

1 <x-layout title="Criar Aluno"> rota que o form será enviado
2 <form action="/alunos/create" method="POST">
3   @csrf diretiva de segurança
4
5   <div class="mb-3"> campos
6     <label for="nm_aluno" class="form-label">Nome Aluno</label>
7     <input class="form-control" name="nm_aluno" type="text" id="nm_aluno" placeholder="Nome do aluno" required>
8   </div>
9   <div class="mb-3">
10    <label for="nm_curso" class="form-label">Nome Curso</label>
11    <input class="form-control" name="nm_curso" type="text" id="nm_curso" placeholder="Nome do curso" required>
12  </div>
13  <div class="mb-3">
14    <label for="nu_ano" class="form-label">Ano do curso</label>
15    <input class="form-control" name="nu_ano" type="number" id="nu_ano" placeholder="2020" required>
16  </div>
17  <div class="mb-3">
18    <label for="nu_semestre" class="form-label">Número do semestre atual</label>
19    <input class="form-control" name="nu_semestre" type="number" id="nu_semestre" max="10" placeholder="1" required>
20  </div>
21
22  <button class="btn btn-dark" type="submit">Cadastrar Aluno</button> botão de envio
23 </form>
24 </x-layout>

```

→ **Interface Visual**

Estilizamos os campos e colocamos um placeholder para facilitar a experiência do usuário. O botão “Cadastrar Aluno” faz o formulário ser enviado chamando a rota selecionada.

2. Definição da rota POST para apontar para o Controller e função dedicada.

```
11 Route::get('/alunos', [AlunosController::class, 'index']);
12 Route::get('/alunos/create', [AlunosController::class, 'create']);
13
14 Route::post('/alunos/create', [AlunosController::class, 'insert']);
```

**Perceba que mesmo utilizando duas rotas com URLs Idênticas (/alunos/create) as rotas chamam diferentes funções, pois possuem verbos HTTP DIFERENTES.**

**NOTA:** Quando vamos lidar com uma requisição que recebe um corpo (body), como chave e valor dos campos de um formulário, utilizamos o verbo POST.

→ Note também que, quando somente queremos retornar algo, como uma view, simplesmente utilizamos o método GET.

→ Entender a diferença entre a utilização destes diferentes verbos é muito importante para compreender o real funcionamento de uma aplicação.

3. Definição da função no Controller para lidar com a requisição.

```
21 public function create()
22 {
23     return view('alunos.create');
24 }
25
26 public function insert(Request $request)
27 {
28     $aluno = new Aluno();
29
30     $aluno->nm_aluno = $request->input('nm_aluno');
31     $aluno->nm_curso = $request->input('nm_curso');
32     $aluno->nu_ano = $request->input('nu_ano');
33     $aluno->nu_semestre = $request->input('nu_semestre');
34
35     $aluno->save();
36
37     return redirect('alunos');
38 }
39 }
```

Retorna view com o formulário

Recebe como parâmetro o valor dos campos do formulário

Instancia um novo aluno

Atribui a instância o valor de cada atributo vindo do formulário

Utiliza função save do Eloquent para salvar a instância do modelo ao Banco de Dados

Redireciona o usuário para a página de listagem de alunos

Na função “insert”, recolhemos a requisição do usuário, que são os campos do formulário, atribuímos a uma nova instância de Aluno, salvamos no banco, e por fim, redirecionamos o usuário para a página de listagem de alunos.

**Para mais informações:** [Doc. Oficial de HTTP Request no Laravel](#)

## **6. Fixando o conhecimento: Desafio**

**Finalize o CRUD da aplicação de Alunos trabalhada ao longo da apostila desta semana.**

**IMPORTANTE:** Para que você possa recuperar os códigos você deve acessar o repositório:

→ <https://github.com/RafaelR4mos/php-laravel-projeto-alunos-exemplo/tree/feat/inserir-aluno>

Há duas formas de recuperar os códigos

7. Clonando o repositório.

8. Baixando o zip dos arquivos.

**LEMBRE-SE de configurar o projeto. Você precisa criar um novo arquivo .env com a sua conexão do banco de dados.**

### **Requisitos do desafio**

- Renderizar um botão de “Deletar Aluno” na listagem dos alunos.
- Criar uma rota de verbo “DELETE” que recebe um parâmetro (id\_aluno).
- Criar uma função no “AlunosController” chamada “destroy” que deve deletar o modelo de aluno específico do banco de dados
  - Se a remoção der certo, você deve redirecionar o usuário para a página de listagem de alunos
  - Se a remoção NÃO der certo, você deve retornar uma resposta JSON com um erro e um status code adequado.
- Renderizar um botão de “Editar Aluno”, na listagem dos alunos.
- Ao acessar o formulário de edição de Aluno ele deve estar com os campos pré-preenchidos com as informações atuais do Aluno para o usuário decidir ou não deletar.
- Criar uma rota com de verbo “PUT”, que deve fazer a edição deste aluno com os novos dados editados.
- Criar uma função no “AlunosController” chamada “update”, que deve Editar o aluno com as novas informações no banco de dados.
  - Caso a edição ocorra da forma esperada, você deve redirecionar o usuário para a página de listagem de alunos.
  - Caso a edição NÃO der certo, você deve retornar uma resposta JSON com um erro e o statusCode adequados.

**\* NÃO HESITE EM ACESSAR A [DOCUMENTAÇÃO OFICIAL DO LARAVEL](#).**

→ Solução do desafio: [Repositório da solução do desafio](#).

## 7. Referências

Múltiplos autores: *Bootstrap: Get started with Bootstrap*, não informado. Disponível em:

<<https://getbootstrap.com/docs/5.3/getting-started/introduction/>>. Acesso em: 12 de novembro de 2023.

Múltiplos autores: Laravel: *Installation*, não informado. Disponível em:

<<https://laravel.com/docs/10.x>>. Acesso em: 12 de novembro de 2023.

Múltiplos autores: *Facades*, não informado. Disponível em <<https://laravel.com/docs/10.x/facades>>. Acesso em 16 de novembro de 2023.

Múltiplos autores: *Creating Tables*, não informado. Disponível em

<<https://laravel.com/docs/10.x/migrations#creating-tables>>. Acesso em 16 de novembro de 2023.

Múltiplos autores: *Available Column types*, não informado. Disponível em

<<https://laravel.com/docs/10.x/migrations#available-column-types>>. Acesso em 16 de novembro de 2023.

Múltiplos autores: *Database: Migrations*, não informado. Disponível em

<<https://laravel.com/docs/10.x/migrations>>. Acesso em 17 de novembro de 2023.

Múltiplos autores: *Configuration*, não informado. Disponível em

<<https://laravel.com/docs/10.x/database#configuration>>. Acesso em 17 de novembro de 2023.

Múltiplos autores: *Eloquent Model Conventions*, não informado. Disponível em

<<https://laravel.com/docs/10.x/eloquent#eloquent-model-conventions>>. Acesso em 17 de novembro de 2023.

Múltiplos autores: *HTTP Requests*, não informado. Disponível em <<https://laravel.com/docs/10.x/requests>>.

Acesso em 19 de novembro de 2023.