1. Migrations

database/migrations

Através do Laravel também é possível criar e modificar tabelas e campos lá no nosso banco de dados.

Criar as tabelas através do migration garante que para onde você levar o código do seu projeto você também estará levando a estrutura do banco de dados que ele precisa.

Não será preciso criar as tabelas e campos através da interface de gerenciamento do MySql.
Na verdade, usando o migration não importa muito qual o banco de dados que está utilizando.
Ele possui uma linguagem própria para criar e modificar os campos, tipos e tamanhos.

Antes de criar uma migração precisamos ter em mente se o objetivo será "criar uma tabela" ou "modificar os campos de uma tabela que já existe".

Para criar um arquivo de migração que vai adicionar uma tabela no banco de dados basta rodar o seguinte comando dentro do Terminal (dentro da pasta do projeto):

```
php artisan make:migration criar_tabela_users --create=users
```

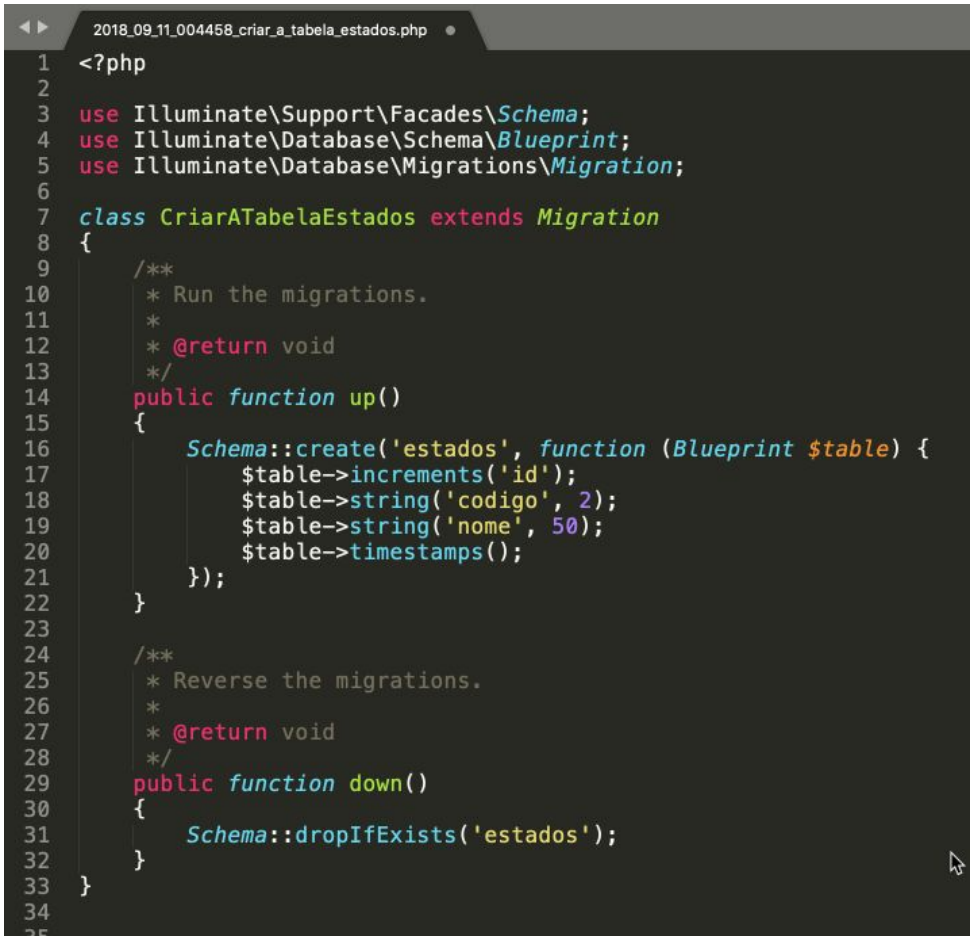
E para criar um arquivo de migração que vai modificar os campos de uma tabela basta:

```
php artisan make:migration adicionar_nota_em_users --table=users
```

Para seguir as boas práticas, o nome do arquivo deverá possuir sempre letras minúsculas.
Também deverá possuir de forma bastante literal qual a ação que ele executará no banco de dados, separando tudo com *underline* ("_").

Utilizando esse comando o arquivo de migração será criado dentro da pasta "`database/migrations`".
E automaticamente o Laravel coloca a data e hora de criação como prefixo no nome do arquivo.

Abaixo um exemplo do arquivo de migração chamado "`criar_a_tabela_estados`":



Toda migração possui duas *public function*:

```
// Executa o que a migração deve fazer (criar ou alterar uma tabela)
public function up() {}

// Desfaz o que a migração fez (apaga a tabela ou desfaz a alteração no campo)
public function up() {}
```

Veja que a função "`up`" está criando uma tabela chamada "`estados`" no banco de dados.
E nessa tabela deverá existir:

- Um campo chamado "`id`", que será incrementado automaticamente;
- Um campo chamado "`codigo`", do tipo string e com tamanho 2;
- Um campo chamado "`nome`", do tipo string e com tamanho 50;
- E também os campos responsáveis por armazenar o timestamp da inclusão e da última atualização ("`created_at`" e "`updated_at`").

Para conferir todos os métodos que podemos utilizar aqui basta conferir a documentação do Laravel: https://laravel.com/docs/5.7/migrations#columns

Depois que criou o arquivo de migração e programou exatamente o que ele precisa fazer você só precisa rodar um comando do artisan para que o nosso Framework execute tudo no banco de dados.

Dentro da pasta do projeto:

```
// Executa todas as funções UP das migrações que não foram executadas ainda
php artisan migrate

// Exibe uma tabela com a situação de todas as migrações
php artisan migrate:status

// Desfaz as últimas migrações executadas (executa as funções DOWN)
php artisan migrate:rollback

// Desfaz todas as migrações executadas (executa as funções DOWN)
php artisan migrate:reset

// Desfaz tudo e executa tudo de novo =)
php artisan migrate:refresh
```

Se por algum acaso criou uma migração com o nome errado, ou não quer mais aquele arquivo no projeto, basta excluir ele de dentro da pasta =)

2. Seeders

database/seeds

Utilizado para popular informações nas tabelas que criou.

Algumas tabelas do sistema podem precisar de registros iniciais dentro do projeto.
Trata-se de tabelas que não terão uma tela de cadastro.
Ou até mesmo tabelas mais estáticas como "Status do Pedido" ou "Estados do Brasil".

O Laravel permite criar uma classe que será responsável por preencher determinada tabela do banco de dados com registros.

Para criar esse arquivo basta:

```
php artisan make:seeder EstadoSeeder
```

Atenção para o padrão que vamos utilizar no nome do seeder.
Começar com letra maiúscula e colocar o sufixo "Seeder" no nome do arquivo, ok?

O arquivo será criado sempre com uma função "`run`":

```php
<?php

use Illuminate\Database\Seeder;

class EstadosSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //

    }
}
```

E tudo que precisamos fazer é incluir o código para cadastrar os registros que queremos.

Para cadastrar alguns estados do Brasil, por exemplo, podemos usar o modelo de estados e o método `create` - igual ao que utilizamos no controlador, lembra?

```php
<?php

use Illuminate\Database\Seeder;
use App\Estado;

class EstadosSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $estado = Estado::create([
            'codigo' => 'AM',
            'nome' => 'Amazonas'
        ]);
        $estado->save();

        $estado = Estado::create([
            'codigo' => 'AP',
            'nome' => 'Amapá'
        ]);
        $estado->save();

    }
}

```

Depois de criar o arquivo e colocar o código para alimentar a tabela com alguns registros, basta rodar o seguinte comando para o Laravel executar isso tudo no banco de dados:

```
// Executa todos os arquivos seeders e alimenta as tabelas do banco de dados
php artisan db:seed
```

Pronto.
Agora é só conferir no banco de dados se as tabelas possuem estes novos registros.

3. Factories

database/factories

Trata-se de uma "receita de bolo" de como o Laravel poderá produzir ou criar dados e informações de fake para popular uma tabela do seu banco de dados.

Facilita muito o dia a dia do programador que precisa conferir se o sistema vai conseguir lidar com grandes volumes de dados.
Ou até mesmo para fazer uma demonstração ou apresentação do sistema para outros.

Muitos utilizam esse recurso apenas para testes mesmo =)

Primeiro precisamos criar um arquivo que vai ter essa "receita de bolo".
Assim:

```
php artisan make:factory EstadoFactory --model=Estado
```

O arquivo com a factory será criado em `database/factories`.
Ele deve ficar mais ou menos assim:

```php
<?php

use Faker\Generator as Faker;

$factory->define(App\Estado::class, function (Faker $faker) {
    return [
    ];
});
```

Agora só temos que preencher, dentro do return, os campos que queremos alimentar com dados fake e de que forma o Laravel precisa "inventar" as informações.

Para saber todas as possibilidades de dados de "mentira" que ele consegue criar basta conferir a documentação do Faker: https://github.com/fzaninotto/Faker

Olha esse exemplo:

```php
<?php

use Faker\Generator as Faker;

$factory->define(App\Estado::class, function (Faker $faker) {
    return [
        'nome' => $faker->name,
        'codigo' => $faker->randomElement($array = array ('AA','BB','CC'))
    ];
});
```

Esse código diz para o Laravel que o campo "`nome`" deve ter inventar um "`name`".
E o campo "`codigo`" deve ser alimentado de forma aleatória com "AA", "BB" ou "CC", por exemplo.

Depois de criar o arquivo com a factory, que é a nossa "receita de bolo", só precisamos mudar um pouco o nosso seeder.

Ele fica mais ou menos assim:

```php
<?php

use Illuminate\Database\Seeder;
use App\Estado;

class EstadosSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {

        factory(App\Estado::class)->times(50)->create();

    }
}
```

Essa linha de código dentro da function `run` vai carregar a nossa "receita de bolo" e criar rapidamente 50 novos estados na tabela do banco de dados!