Database: Seeding

Introduction

Laravel includes a simple method of seeding your database with test data using seed classes. All seed classes are stored in `database/seeds`. Seed classes may have any name you wish, but probably should follow some sensible convention, such as `UserTableSeeder`, etc. By default, a `DatabaseSeeder` class is defined for you. From this class, you may use the `call` method to run other seed classes, allowing you to control the seeding order.

Writing Seeders

To generate a seeder, you may issue the `make:seeder` Artisan command. All seeders generated by the framework will be placed in the `database/seeders` directory:

```
php artisan make:seeder UserTableSeeder
```

A seeder class only contains one method by default: `run`. This method is called when the `db:seed` Artisan command is executed. Within the `run` method, you may insert data into your database however you wish. You may use the query builder to manually insert data or you may use Eloquent model factories.

As an example, let's modify the `DatabaseSeeder` class which is included with a default installation of Laravel. Let's add a database insert statement to the `run` method:

```
<?php

use DB;
use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
}
```

Using Model Factories

Of course, manually specifying the attributes for each model seed is cumbersome. Instead, you can use model factories to conveniently generate large amounts of database records. First, review the model factory documentation to learn how to define your factories. Once you have defined your factories, you may use the `factory` helper function to insert records into your database.

For example, let's create 50 users and attach a relationship to each user:

```
/**
 * Run the database seeds.
 *
 * @return void
 */
public function run()
{
    factory(App\User::class, 50)->create()->each(function($u) {
        $u->posts()->save(factory(App\Post::class)->make());
    });
}
```

Calling Additional Seeders

Within the `DatabaseSeeder` class, you may use the `call` method to execute additional seed classes. Using the `call` method allows you to break up your database seeding into multiple files so that no single seeder class becomes overwhelmingly large. Simply pass the name of the seeder class you wish to run:

```
/**
 * Run the database seeds.
 *
 * @return void
 */
public function run()
{
    Model::unguard();

    $this->call(UserTableSeeder::class);
    $this->call(PostsTableSeeder::class);
    $this->call(CommentsTableSeeder::class);
}
```

Running Seeders

Once you have written your seeder classes, you may use the `db:seed` Artisan command to seed

your database. By default, the `db:seed` command runs the `DatabaseSeeder` class, which may be used to call other seed classes. However, you may use the `--class` option to specify a specific seeder class to run individually:

```
php artisan db:seed

php artisan db:seed --class=UserTableSeeder
```

You may also seed your database using the `migrate:refresh` command, which will also rollback and re-run all of your migrations. This command is useful for completely re-building your database:

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