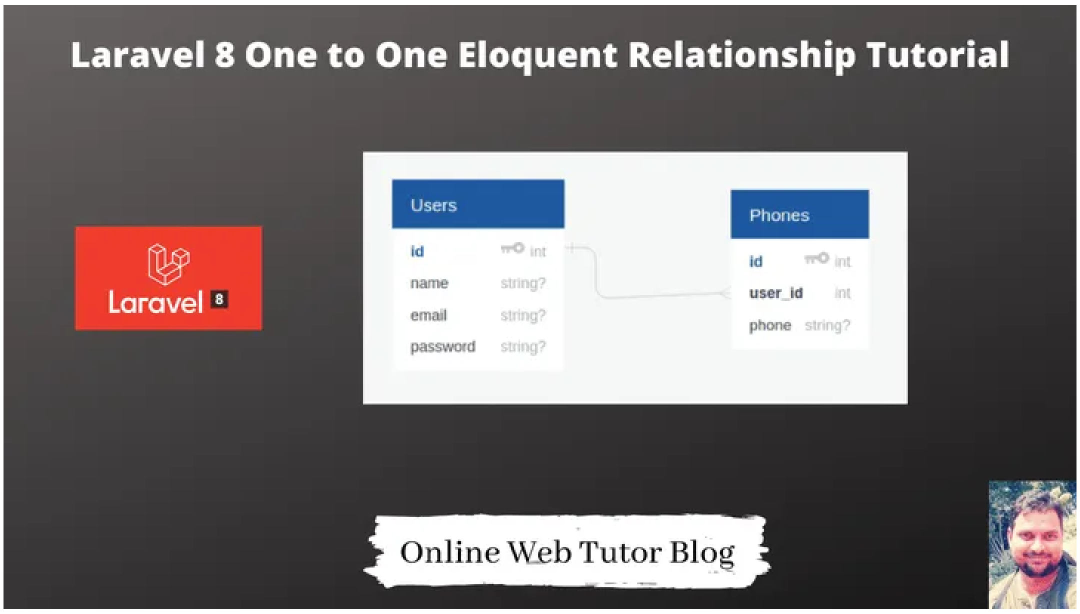
34% OFF on Premium Library Management System WordPress Plugin. Buy Now



Laravel 8 One to One Eloquent Relationship Tutorial

March 31, 2021 by Sanjay Kumar

Table of Contents

Share this Article

Laravel eloquent relationship is a very important feature which connects one or more tables in a chain. This is the substitute of joins in laravel.

Laravel provides these following relationships –

- **One To One**
- **One To Many**
- **Many To Many**
- One To Many (Inverse) / Belongs To
- **Has One Through**
- **Has Many Through**

Eloquent relationships are defined as methods on your Eloquent model classes. Inside this article we will see the concept of laravel 8 One to One Eloquent relationship as well as we will implement inverse of one to one relationship i.e belongs to.

This article will give you the detailed concept of about implementation of one to one relationship in laravel.

For this tutorial we will consider a **users table** and a **phones table**. This means a single user has a single phone number.

Let’s get started.

Installation of Laravel Application

Laravel Installation can be done in two ways.

- Laravel Installer
- By using composer

Laravel Installer

To install Laravel via Laravel installer, we need to install it’s installer first. We need to make use of composer for that.

```
$ composer global require laravel/installer
```

This command will install laravel installer at system. This installation is at global scope, so you type command from any directory at terminal. To verify type the given command –

```
$ laravel
```

This command will open a command palette of Laravel Installer.

To create ad install laravel project in system,

```
$ laravel new blog
```

With the name of **blog** a laravel project will be created at your specified path.

By using composer

If your system doesn't has composer installed, **Learn Composer Installation Steps**.

Here is the complete command to create a laravel project-

```
$ composer create-project --prefer-dist laravel/laravel blog
```

After following these steps we can install a Laravel application into system.

To start the development server of Laravel –

```
$ php artisan serve
```

This command outputs –

Starting Laravel development server: http://127.0.0.1:8000

Assuming laravel already installed at system.

Create Database & Connect

To create a database, either we can create via Manual tool of PhpMyadmin or by means of a mysql command.

```
CREATE DATABASE laravel_app;
```

To connect database with application, Open **.env file** from application root. Search for **DB_** and update your details.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_app
DB_USERNAME=root
DB_PASSWORD=root
```

Create Migrations

We will find migration **2014_10_12_000000_create_users_table.php** of users at /database/migrations.

Open up the migration file, we should see this following code into it.

#Users

```
<?php

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Open project in terminal and run this spark command to **create migration for phones** table.

```
$ php artisan make:migration CreatePhonesTable
```

It will create a file 2021_03_31_170942_create_phones_table.php at **/database/migrations** according to the timestamp value.

Open file and update code with this code.

#Phones

```
<?php
```

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

class CreatePhonesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('phones', function (Blueprint $table) {
            $table->id();
            $table->integer('user_id')->unsigned();
            $table->string('phone', 30);
            $table->foreign('user_id')
                ->references('id')->on('users')
                ->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('phones');
    }
}
```

Run Migrations

Next, we need to create tables inside database.

```
$ php artisan migrate
```

This command will create tables inside database.

Create Model

We need to create few models inside application. By default **User.php** is available inside application setup.

Open User.php and update with this code.

User.php

```
namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use App\Models\Phone;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne(Phone::class);
        // OR return $this->hasOne('App\Phone');
    }
}
```

hasOne() method is used to get associated phone number of user. This is one to one relationship where each user h
been associated to specific mobile number.

We will create **Phone.php** model.

```
$ php artisan make:model Phone
```

It will create **Phone.php** file at /app/Models folder. Open file and write this code.

Phone.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\User;

class Phone extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'user_id',
        'phone'
    ];

    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo(User::class);
        // OR return $this->belongsTo('App\User');
    }
}
```

belongsTo() method is implementing inverse relationship of one to one relationship in laravel.

Open project into terminal and type these artisan command.

```
$ php artisan make:factory UserFactory --model=User

$ php artisan make:factory PhoneFactory --model=Phone
```

It will create two files, **UserFactory.php** and **PhoneFactory.php** at /database/factories folder.

Open **UserFactory.php** file and write this code into it.

UserFactory.php

```php
<?php

namespace Database\Factories;

use App\Models\User;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = User::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name,
            'email' => $this->faker->unique()->safeEmail,
            'email_verified_at' => now(),
            'password' => bcrypt("123456"), // password
            'remember_token' => Str::random(10),
        ];
    }
}
```

Open **PhoneFactory.php** file and write this code.

PhoneFactory.php

```php
<?php
```

```
use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\User;
use App\Models\Phone;

class PhoneFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Phone::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            "user_id" => \App\Models\User::factory()->create()->id,
            "phone" => $this->faker->phoneNumber
        ];
    }
}
```

Here, we are generating data using One to One Eloquent relationship.

Open terminal and type

```
$ php artisan tinker
```

Inside tinker shell panel, run this command to seed dummy data into database table.

```
>>> App\Models\Phone::factory()->count(10)->create()
```

This command will seed data into users table and phones table as well. In both tables we will have 10 number of records.

Calling Methods at Controller

Open any controller say **SiteController.php** file, we have created two methods in which we used model methods as a property.

```
#Controller
<?php

namespace App\Http\Controllers;
```

```
use App\Models\Phone;

class SiteController extends Controller
{
    public function getPhone($user_id)
    {
        // Passing user id into find()
        return User::find($user_id)->phone;
    }

    public function getUser($phone_id)
    {
        // Passing phone id into find()
        return Phone::find($phone_id)->user;
    }
}
```

- **User::find($user_id)->phone;** It will find phone details value by user id. **One to One**
- **Phone::find($phone_id)->user;** It will find user details by phone id. **Inverse of One to One / Belongs To**

Create Routes

Open **web.php** from /routes folder and add these routes into it.

web.php

```
# Add this to header
use App\Http\Controllers\SiteController;

Route::get('get-phone/{id}', [SiteController::class, 'getPhone']);
Route::get('get-user/{id}', [SiteController::class, 'getUser']);
```

Application Testing

Open project to terminal and type the command to start development server

```
$ php artisan serve
```

Get Phone details – http://127.0.0.1:8000/get-phone/1

Get User details – http://127.0.0.1:8000/get-user/1

We hope this article helped you to learn about Laravel 8 One to One Eloquent Relationship Tutorial in a very detailed way.

If you liked this article, then please subscribe to our **YouTube Channel** for PHP & it's framework, WordPress, Node Js video tutorials. You can also find us on **Twitter** and **Facebook**.

Find More on Laravel 8 Articles here

- How to Create Multi Language Website in Laravel 8

- Laravel 8 Ajax Post Request Tutorial
- Laravel 8 Authentication using Jetstream with Inertia Js
- Laravel 8 Authentication using Jetstream with Livewire
- Laravel 8 Authentication with Breeze Tutorial
- Laravel 8 Clear Cache of Route, View & Config
- Laravel 8 Cron Job Task Scheduling Tutorial
- Laravel 8 DataTable Ajax Pagination with Search And Sort
- Laravel 8 Firebase Push Notification Tutorial
- Laravel 8 Form Validation Methods
- Laravel 8 Installation Guide – PHP Framework
- Laravel 8 Layouts And Views Complete Guide
- Laravel 8 Routing Tutorial Step by Step Guide
- Laravel 8 Send Mail using Gmail SMTP Server
- Laravel 8 Send Push Notification to Android Using Firebase
- Laravel 8 Send Push Notification to IOS Using Firebase
- Laravel 8 Stub Customization

Related Posts:

- Laravel 8 One to Many Eloquent Relationship Tutorial
- Laravel 8 Has One Through Eloquent Relationship Tutorial
- Laravel 8 Has Many Through Eloquent Relationship Tutorial
- Laravel 8 Many to Many Eloquent Relationship Tutorial
- Eloquent Mutators and Accessors in Laravel 8 Tutorial
- Laravel 8 Store Log Of Eloquent SQL Queries

Laravel 8

‹ Multiple Images Upload in CodeIgniter 4 Tutorial

› Laravel 8 One to Many Eloquent Relationship Tutorial

Advertisement

Menu

ezoic report this ad

Find Us on Youtube

Online Web Tutor Youtube

Advertisement

Menu

ezoic report this ad

Popular Posts

Sorry. No data so far.

Categories

- Blogging (1)
- CakePHP 4 (11)
- CodeIgniter 4 (156)
- Education (1)
- jQuery & Javascript (39)
- Laravel 8 (152)
- MySQL (10)
- Node Js (15)
- PHP Miscellaneous (33)
- Wordpress (22)

Menu

ezoic report this ad

Recent Posts

How To Read XML Data to JSON in CodeIgniter 4

How To Read XML Data to JSON in PHP Tutorial

MySQL Sum with If Condition Tutorial

What are Working Components of CodeIgniter 4?

Concept of Lazy Loading Image Using jQuery

ezoic report this ad

Menu

Menu

Menu

Menu

Menu

© 2022 Online Web Tutor