MENU

Creando un API RESTful con Laravel 5.2 parte 1

16 FEBRUARY 2016

Vamos a iniciar una serie de tutoriales en Laravel 5.2. Mi nombre es Martín Guadalupe y soy desarrollador en CoffeeDevs. En esta oportunidad comenzaré explicando

paso a paso como construir una API RESTful muy sencilla utilizando los componentes más importantes de este increíble framework.

Lo primero que necesitamos para este tutorial, es tener instalado composer en nuestro sistema.

En caso de Linux

```
curl -sS https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
```

En caso de Windows
Descargar Composer

Para mas información
https://getcomposer.org/download/

Instalación de Laravel

Una vez instalado composer, podemos crear un proyecto de dos maneras:

Usando el instalador de Laravel

Necesitamos descargar el instalador con el comando

```
composer global require "laravel/installer=~1.1"
```

Una vez descargado, tenemos que agregar el directorio ~/.composer/vendor/bin al path del sistema operativo,

usando Linux. En caso de Windows el directorio suele estar en la ruta C:\Users{miUsuario}\AppData\Roaming\Composer\vendor\bin

Para instalar nuestro proyecto usaremos el comando:

```
laravel new todolist
```

Usando Composer

```
composer create-project laravel/laravel todolist --prefer-dist
```

Ya con el proyecto creado procederemos a crear las bases para nuestra aplicación.

Lo primero que vamos a hacer es copiar el archivo ".env.example" con el nombre ".env".
Cada aplicación Laravel necesita este archivo para poder obtener la información local necesaria de los diferentes componentes del framework.
Para almacenar los datos utilizaremos MySql como gestor de base de datos. Necesitaremos crear una base de datos nueva a la que llamaremos "laravel_todolist".

Ya creada la base de datos, procederemos a modificar el archivo .env por defecto con los valores de su sistema. Una posible configuración sería:

```
DB_HOST=localhost
DB_DATABASE=laravel_todolist
DB_USERNAME=root
DB_PASSWORD=
```

Migraciones

Las migraciones funcionan como un sistema de control de versiones para base de datos.
Aquí crearemos el esquema de nuestra base.

`php artisan make:migration create_projects_table --create=projects`

Este comando creará una nueva migración dentro de database/migrations.
Ahora deberemos definir los campos de la tabla projects:

```
<?php

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

class CreateProjectsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
```

```
    {
        Schema::create('projects', function(Blueprint
$table)
        {
            $table->increments('id');
            $table->string('name')->default('');
            $table->timestamps();
        });
    }

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

Repetimos lo mismo para la tabla tasks

```
php artisan make:migration create_tasks_table --create=tasks
```

```
<?php

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

```
class CreateTasksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('tasks', function(Blueprint
$table) {
            $table->increments('id');
            $table->integer('project_id')->unsigned()-
>default(0);
            $table->foreign('project_id')-
>references('id')->on('projects')->onDelete('cascade');
            $table->string('name')->default('');
            $table->boolean('completed')->default(false);
            $table->text('description')->default('');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
```

```
        Schema::drop('tasks');
    }
}
```

Luego ejecutamos `php artisan migrate`
Así nuestra migracion se llevará a cabo.

Models

Necesitaremos crear los modelos correspondientes a cada tabla.

```
php artisan make:model Project
php artisan make:model Task
```

Para poder guardar nuevos datos, tendremos que configurar los modelos.

```
// app/Project.php
class Project extends Model
{
    protected $fillable = ['name'];
}
```

```
// app/Task.php
class Task extends Model
{
    protected $fillable =
['project_id','name','completed','description'];
}
```

Seeders

Laravel nos provee un conjunto de clases que nos permiten llenar con datos de prueba nuestra base de datos.
Con artisan podemos crear estas clases fácilmente:

```
php artisan make:seeder ProjectsSeeder
php artisan make:seeder TasksSeeder
```

Estos comandos crearán las clases ProjectsSeeder y TasksSeeder dentro de la carpeta database/seeds.

Necesitaremos llamarlos desde la clase DatabaseSeeder ubicada también en database/seeds.

```
Model::unguard();

$this->call(ProjectsSeeder::class);
$this->call(TasksSeeder::class);

Model::reguard();
```

Ahora escribiremos en cada clase para crear los datos de prueba.

```
class ProjectsSeeder extends Seeder
{
    public function run()
    {
        Project::create([
```

```
            'id' => 1,
            'name' => 'Project 1',
        ]);
        Project::create([
            'id' => 2,
            'name' => 'Project 2',
        ]);
    }
}
```

```
class TasksSeeder extends Seeder
{
    public function run()
    {
        Task::create([
            'project_id' => 1,
            'name' => 'Task 1',
        ]);

        Task::create([
            'project_id' => 2,
            'name' => 'Task 1',
        ]);

        Task::create([
            'project_id' => 2,
            'name' => 'Task 2',
        ]);
    }
}
```

Para volcar estos datos en las tablas projects y tasks usaremos el siguiente comando:

```
php artisan db:seed
```

Si ocurre algún error podemos utilizar el comando:

```
composer dump-autoload.
```

Routes

Para definir las diferentes rutas que va a tener nuestra API tendremos que escribirlas en app/Http/routes.php
La configuración será la siguiente:

```
<?php
Route::resource('projects', 'ProjectController');
Route::resource('projects.tasks',
'ProjectTaskController');
```

Definimos Route::resource('projects','ProjectController') para indicarle a Laravel que que vamos a utilizar las rutas preparadas para RESTful.

Controladores

Definimos los controladores llamados en las rutas, dentro de app/Http/Controllers. Vamos a crear el primer endpoint para obtener un listado del recurso solicitado. En este caso miproyecto/projects por un lado y miproyecto/projects/{projectId}/tasks por el otro.

Para crear los controladores mediante artisan usaremos:

```
php artisan make:controller ProjectController
php artisan make:controller ProjectTaskController
```

```
<?php
// app/Http/Controllers/ProjectController.php
namespace App\Http\Controllers;

use App\Project;
use App\Http\Requests;
use Illuminate\Support\Facades\Input;

class ProjectsController extends Controller
{
    public function index()
    {
        return Project::all();
    }
}
```

```
<?php
// app/Http/Controllers/Tasks

namespace App\Http\Controllers;

use App\Project;
use App\Task;
use App\Http\Requests;
```

```
use Illuminate\Support\Facades\Input;

class ProjectTasksController extends Controller
{
    public function index($projectId)
    {
        return Task::where('project_id',$projectId)-
>get();
    }
}
```

Teniendo tanto las rutas como los controladores, ya podemos consumir esta sencilla API RESTful. Para poder acceder al listado de proyectos debemos iniciar el servidor usando el comando php artisan serve. Una vez iniciado, podemos acceder a los proyectos utilizando las siguientes rutas:

http://localhost:8000/projects
Nos devolverá un json con todos los proyectos disponibles en la base de datos.

http://localhost:8000/projects/{projectId}/tasks
Nos devolverá un json con todas las tareas disponibles para el proyecto con ese id.

En esta parte vimos como crear los componentes básicos de Laravel para construir un API RESTful. En los siguientes tutoriales vamos a completar el API logrando así agregar, eliminar y modificar los diferntes recursos.

Gracias por leernos!

Encontranos en @coffeedevs

Martín Guadalupe

Read more posts by this author.

CABA, Argentina

Share this post

READ THIS NEXT

Un año de DigitalOcean - Nuestra experiencia

Como agencia de desarrollo web, en CoffeeDevs tuvimos que decidir desde el primer día, qué tipo de servicio de...

YOU MIGHT ENJOY

Lista de cambios en español para Laravel 5.2

Laravel 5.2 Laravel 5.2 continua las mejoras realizadas en Laravel 5.1, agregando soporte para múltiples controladores...