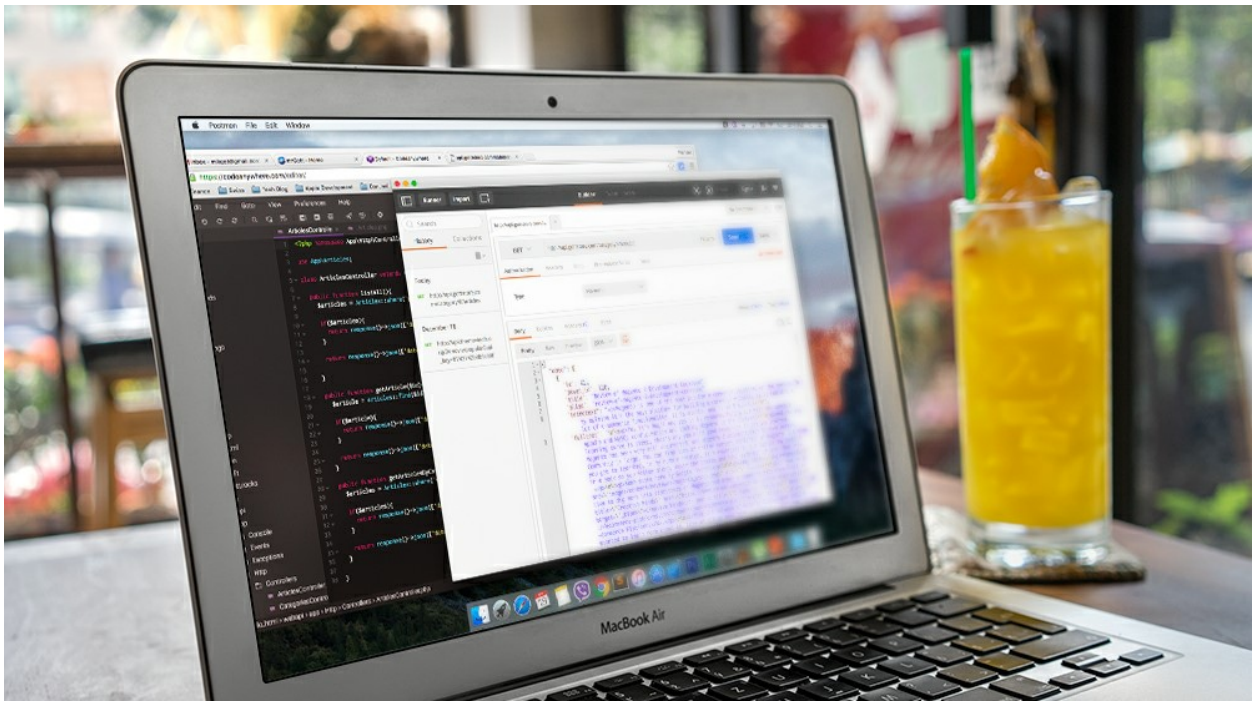
Build Secure PHP RESTful API For Your Application With Lumen 5

Published: 18 April 2016 Category: Joomla



Finally, I managed to write this second article on how to Create Your First RESTful Web Service For Joomla! 3 With Lumen.

For this example we are building a webservice with Lumen for my blog. It's a standard Joomla 3.4 installation. So if you want to follow along you can simply install Joomla 3 or above on your localhost or just take a look at the code and learn how to build a webservice with Lumen 5.

I'll keep in mind that, you, reader of my blog, might not be a php developer, have never worked with php and you just need to create a simple API for your mobile application. The beauty of Lumen is that you can create php based webservice for any php based web application very easily.

To start off, we need to figure out what we are going to build and then make a blueprint of output data.

My blog consists of standard Joomla categories and articles assigned to them. So in this case we'll have one to many connection type. This is why I decided to develop example webservice for my blog.

Things will be very simple. We are going to make only GET calls. It means that our web service will only be returning requested JSON data, and we won't be posting any data back to the server.

From this point I don't see the need of posting data back to the server (implement Create, Update, Delete methods). In case of Joomla, you have a very neat and nice back-end. If you need

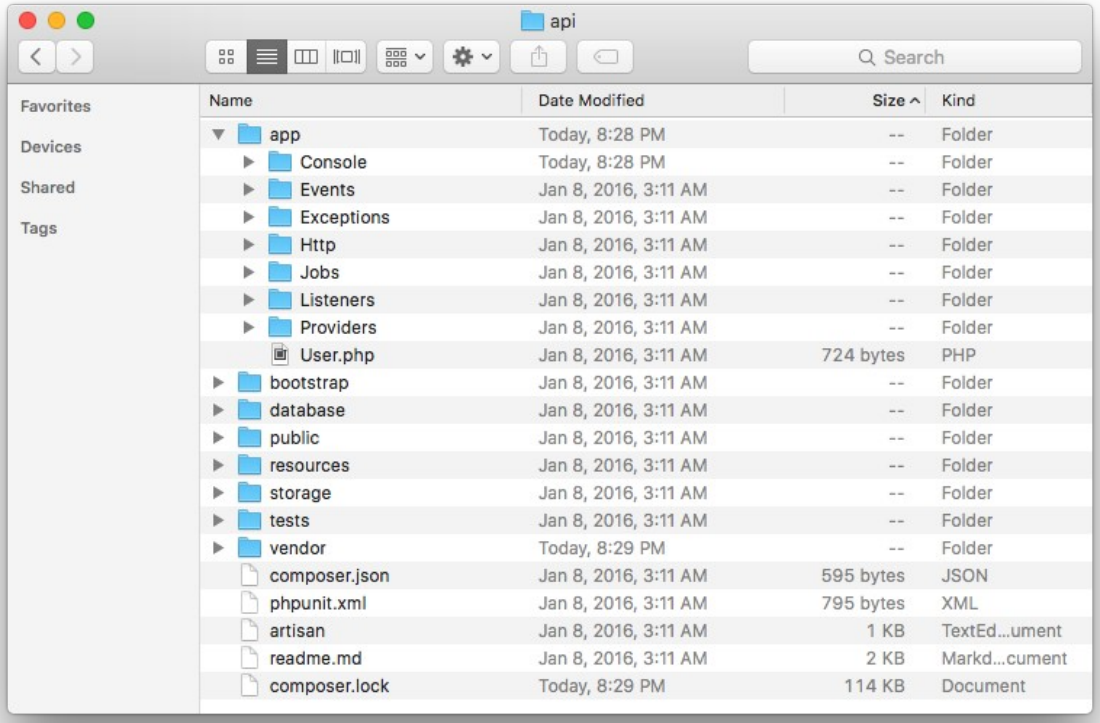
to further extend your API, I would recommend watching video tutorial on Udemy - Create Fast RESTful APIs with Lumen and Homestead by Laravel by JuanD MeGon. Judging by the introductory video, he's English is not perfect but I'm sure code is way more "international" language then English in this case. And the contents are really impressive.

Later we will add Oauth2 token validation and error handling the way I have described it in my article: HTTP Status Codes To Handle Errors in Your API.

To be more precise, we will be getting list of all categories, list of all articles, list of all articles in a specific category and a selected article.

So with this in mind let's start diving into development of web service with Lumen.

Before writing actual code it's good to take a look at file structure and configuration of Lumen.



The first directory to note is the **App** directory. This is a folder where all of your logic and code will be located. This is where the routes, controllers and middlewares are stored. Your models will be directly located in the app folder. Your controllers will be placed in **App -> Http -> Controllers**. Your routes to controller/action will be placed inside the **routes.php** file under **App -> Http** directory.

Bootstrap folder contains **app.php** file to bootstrap the framework and configure auto-loading.

Database directory is where the database migrations and seeders are stored. We'll make use of it when we deal with *OAuth2* and will cover it in more details in a later article.

Public directory is where your public assets are stored. Things like css, javascript and images are stored in this directory. But in case of Lumen you won't be using assets, as far as we use Lumen for only web service purposes. If you plan on developing a website that will use assets then, it's better to use Laravel as it includes more functionality and views, although Lumen has **views** directory by default, we really won't be using it to return JSON data to users.

The **Storage** directory is where logs, sessions and cache files are stored.

Tests folder will contain your code for tests and **Vendor** directory is where all of the dependencies of your app are stored. This is where composer installs the packages that you specified in your **composer.json** file.

Before we start we should configure Lumen. As you install Lumen, inside the root folder you will find **.env.example** file. This is where all configuration for Lumen go. It's better if you copy this file and before renaming it to **.env**

Now let's configure Lumen.

Inside the **.env** file first thing to note is the **APP_KEY**. Lumen's documentation states: *"if the application key is not set, your user encrypted data will not be secure"*. Don't really understand what they mean, but let's do if it's required. This parameter should be 32 characters long. To generate random string, just simply search for "Random string generator". In my case I accessed www.textmechanic.com to generate my key.

If you scroll down the **.env** file you will discover parameters for database configuration. Enter those according to your database. In my case it's as follows:

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=jE4GsJNCgVlEicDhXADnhsu5YxoiZBtO

DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=gottstor_api
DB_USERNAME=gottstor_api
DB_PASSWORD=myDBpassword:)

CACHE_DRIVER=memcached
QUEUE_DRIVER=sync
```

The setup is not over yet.

To make Lumen load this configuration file we need to edit the **bootstrap/app.php** file and uncomment the following line:

```
Dotenv::load(__DIR__.'/../');
```

In order to pass the data to users as JSON, first we need to grab the data from the database. We need to enable **Eloquent** (the Lumen/Laravel ORM library). In order to enable Eloquent we need to uncomment several lines in the **bootstrap/app.php** file:

```
$app->withFacades();
$app->withEloquent();
```

Now as all the setup is done we can move on to creating Models and Controllers for our webservice.

As far as we want to display categories and articles we need to create models and controllers respectively.

The beauty of Lumen is its simplicity!

Let's start by making data model for categories. Navigate to the **app** folder and create **Categories.php** file.

Joomla's categories table is called **somePrefix_categories**. Columns that we need are ID and Title. We will filter categories that are published in the system. For example, I have categories in the system that are not published because I'm not using them. So we don't want to have unpublished categories listed. There's also another column to notice in Joomla's categories table. It's called - **extension**. By this extension type we can tell to which component a specific category is related to. As far as all articles are part of *com_content* component we should filter extension column by com_content. We don't want to list categories that are related to *com_banners* or *com_newsfeeds*.

As far as Lumen uses MVC pattern it's clear that we will be dealing with object-oriented PHP. Each class will make use of Namespaces and Use keywords. If you are not familiar with them, Namespaces - in the broadest definition are a way of encapsulating items. This can be seen as an abstract concept in many places. For example, in any operating system directories serve to group related files, and act as a namespace for the files within them. To find out more visit *php.net* – Namespaces and Use Our model for categories table will look like this:

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Categories extends Model {

  protected $table = 'xjts3_categories';
  protected $primaryKey = 'id';

  //Some additional Model code

}
```

In this file we are making use of Eloquent's Model class and we just let it know that the name of categories' table is '**xjts3_categories**' and the primary key **id**. Nothing else is required.

You could add a function to define one-to-many relationship that is done by `this->hasMany('MODEL_NAME')` but I'm not going to use it as far as there is no need.

Now let's create a controller for categories under **App -> Http -> Controllers** and name it **CategoriesController.php**.

Inside *CategoriesController* class we'll have two methods, so called actions: **listAll()** that will list all published items and **getCategory($id)** where we'll pass a category ID and get that specific category. Getting a specific category in my case won't return any important data. But the same method will be used in the Articles controller that will return the article itself.

This is how it looks like:

```php
<?php namespace App\Http\Controllers;

use App\Categories;

class CategoriesController extends Controller {

  public function listAll(){
    $categories = Categories::where('extension','=','com_content')->where('published','=','1'
)->orderBy('lft', 'ASC')->get();

    if($categories){
      return response()->json(['data' => $categories], 200);
    }

    return response()->json(['data' => 'Categories not found'], 404);
  }

  public function getCategory($id){
    $category = Categories::find($id);

    if($category){
      return response()->json(['data' => $category], 200);
    }

    return response()->json(['data' => 'Category not found'], 404);

  }

}
```

As you can see we defined two methods as described earlier. We make a call to the categories' model (before calling we make sure to "import" it by use keyword). As you might notice setting up queries using Eloquent is very simple. I'm sure you can understand the code by just reading it. Here's a cheat sheet for usage of Eloquent.

In each method, after we run the query, we make a check if the results were grabbed from the database. In case they are grabbed, we make a response in JSON format and pass data variable with the data obtained from database, and a HTTP response code. In case of success it's **200** and in case of failure it's **404**.

Before we proceed to creating a Model and a Controller for articles, let's first make sure that everything we've done so far works.

But where does our user get the data? What URL should they call to access the list of categories?

This is when we need to deal with Routes.

As always, when you need to make a GET request to an API you access some URL with some parameters and get the response. In our case we would like to make a call to something like: `http://api.someurl.com/categories`. This is a URL that I came up with, you can set it up whatever you wish.

In many other frameworks the URLs are automatically generated, as well as you can set them up manually. In Lumen we need to configure routes manually.

To set up our routes we need to edit **routes.php** file that can be found under **App -> Http** directory.

Code that defines a route to list all categories like this:

```
$app->get('/categories','CategoriesController@listAll');
```

If you take a closer look, you will see that we are calling get() method that takes several parameters. First is a string, our URL and the second is a path to a method/action **listAll()** that we defined in our *CategoriesController*.

To define another route to list a specific category by passing an ID, you pass corresponding action **getCategory()** from the *CategoriesController* and in the URL part you add parameter name in curly braces just like this:

```
$app->get('/category/{id}','CategoriesController@getCategory');
```

The final code in the routes.php file looks like this:

```
$app->get('/', function () use ($app) {
    return $app->version();
});

$app->get('/categories', 'CategoriesController@listAll');
$app->get('/category/{id}', 'CategoriesController@getCategory');

$app->get('/articles', 'ArticlesController@listAll');
$app->get('/article/{id}', 'ArticlesController@getArticle');
$app->get('/category/{id}/articles', 'ArticlesController@getArticlesByCategory');
```

Now to test it, you can open up a browser and type in the URLs that we just defined. But I would recommend using Postman, a great tool for working with API's. You can get it as a Chrome extension.

As you can see Lumen rocks! This is a basic example of an API and it takes only few minutes to create it. Before I created my first web service, I thought it was a mysterious ancient animal that would live in the dark woods. But with Lumen, it's like a breeze on a sea shore on a hot summer day.

So now let's get back to articles. The steps are very much the same as we did with categories, the only thing different here is a method called **'getArticlesByCategory'**. So I'll directly move on to it.

But first let's create a **Model** for our articles:

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Articles extends Model {

  protected $table = 'xjts3_content';
  protected $primaryKey = 'id';

  //Some additional Model code
```

```
}
```

Now, let's create the **Controller**:

```
<?php namespace App\Http\Controllers;

use App\Articles;

class ArticlesController extends Controller {

  public function listAll(){
    $articles = Articles::where('state','=','1')->get();

    if($articles){
      return response()->json(['data' => $articles], 200);
    }

    return response()->json(['data' => 'Articles not found'], 404);

  }

  public function getArticle($id){
    $article = Articles::find($id);

    if($article){
      return response()->json(['data' => $article], 200);
    }

    return response()->json(['data' => 'Article can not be found'], 404);
  }

  public function getArticlesByCategory($id){
    $articles = Articles::where('catid','=', $id)->where('state','=','1')->orderBy('created',
'DESC')->get();

    if($articles){
      return response()->json(['data' => $articles], 200);
    }

    return response()->json(['data' => 'Articles in this category not found'], 404);
  }

}
```

In fact there's nothing special with the method **'getArticlesByCategory'**. We just filter articles by **category id**. User will pass category id as a parameter into the URL and we will pass it to our query. That's it!

NOTE: *The only thing that I don't love so far is how we get* ***images*** *from Joomla. Joomla stores images inside images column in the* ***com_content*** *table. You might guess that it's a JSON data. I'm not sure how it will be received by user's application. You might need to loop through the array retrieved for database, run php's* ***json_decode*** *on the image and then re-assemble array and then pass it to the user as JSON. Most likely I'll cover it in the series of* ***Reading JSON format data from a remote source in Swift 2 and iOS9****, if I have a chance to write it someday.*

Refactor our code

There's some refactoring I'd like to mention. In the controllers, when returning JSON responses we used the same code. I'm talking about:

```
return response()->json(['data' => $articles], 200); // in case of success

return response()->json(['data' => 'Articles in this category not found'], 404); // in case o
f failure
```

We can move this code to **Controller.php** file inside our *Controllers*' directory like this:

```
<?php

namespace App\Http\Controllers;

use Laravel\Lumen\Routing\Controller as BaseController;

class Controller extends BaseController
{
    //


  public function successResponse($data, $code){
    return response()->json(['data' => $data], $code);
  }

  public function failureResponse($data, $code){
    return response()->json(['data' => $data], $code);
  }

}
```

And inside the controller methods we make a call to this newly created functions from the controllers's methods respectively:

```
  public function listAll(){
    $articles = Articles::where('state','=','1')->get();

    if($articles){
      return $this->successResponse($articles, '200');
    }

    return $this->failureResponse('Articles not found', 404);

  }
```

Conclusion

That's it! Lumen is a breeze! It's very simple, fast and lightweight. Developing a basic API using Lumen is the best thing that can happen to you. Hopefully I did explain everything in details so you won't have any trouble with it. In case some areas remain obscure please feel free to reach my by email. I'll try my best to write the following article, about **OAuth2** implementation into

Lumen so that you can restrict access to your API using API Key, as soon as possible. Cheers!

PART 1 - Create Your First RESTful Web Service For Joomla! 3 With Lumen - Introduction

Prev Next

Related Articles

Play System Sound Notification with AudtioToolbox in Swift 4

UPDATE: Modify System Integrity Protection in El Capitan to Install Mcrypt and Intl Php Extensions

Building a Barcode and a QR Code Reader Application for iOS using Swift 4: PT3

Integrate OAuth2 Server into Lumen to secure your RESTful API with Access Tokens

Getting Started With CocoaPods – Install Your First Dependency: Firebase 3

Installing latest version of Magento 2 1.0.0-beta

Building a Barcode and QR Code Reader Application for iOS using Swift 4: PT1

Firebase Storage - How to download files using Firebase 3 SDK with Swift 3 in Xcode 8

Building a Barcode and a QR Code Reader Application for iOS using Swift 4: PT2

Firebase 3 and Xcode 8 - Hide unnecessary console log output

Categories

Home

Reviews

Swift

Misc

Popular Articles

REST API in Swift 4 using URLSession and JSONDecoder

Create Markdown file and push it to GitHub for The Data Scientist's Toolbox project at Coursera

Install mcrypt and intl php extensions on Mac OS X and XAMPP

Work with REST API in Swift 3 and Xcode 8 using URLSession and JSONSerialization

Setting up Virtual Hosts for XAMPP on Mac OS X (El Capitan)