Learning Laravel's Eloquent

Develop amazing data-based applications with Eloquent, the Laravel framework ORM

Francesco Malatesta
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 *Setting Up Our First Project*
- A synopsis of the book’s content
- More information on *Learning Laravel's Eloquent*
About the Author

Francesco Malatesta, born in 1990, is a web developer and a curious enthusiast from Vasto, a wonderful city on the Italian eastern coast. He actually lives in Rome, where he is studying computer engineering. He received his first PC at the age of six, and since then, they have never separated. It's not all about computers, however; he also likes to travel, add new items to his movie collection, and create new awesome experiences. This book is a perfect example.

"Never stop" is his key.

He is the founder of Laravel-Italia, the Italian Laravel community. He works as a freelancer developer and consultant, but he also writes for Sitepoint, in the PHP section, and HTML.IT, the first portal about information technology in Italy. He started his first job in web development when he was 15 years old.

In the past, he has translated three books, which are Laravel: Code Happy (by Dayle Rees), Code Bright (by Dayle Rees), and Laravel Testing: Decoded (by Jeffrey Way).
If you are associated with the field of web development, you know how important data is. The web runs on data, so it's essential for developers to think of quick and effective ways to deal with it. Eloquent is an awesome ORM that comes with the Laravel PHP framework. It is unique and is very beneficial to developers as it allows them to define models, relationships, and many complex operations with a really easy and intuitive syntax, without sacrificing performance. Performing an interesting number of operations on multiple tables without writing long queries with objects will be a bed of roses.

This book will take you through developing brilliant data-based applications with Eloquent, the Laravel framework ORM.

You will do the following:

- Build highly efficient applications with the Eloquent ORM using an expressive syntax
- Get to grips with the power of relationships and how Eloquent handles them
- Go beyond simple theory with various step-by-step code examples

So, let's get started!

**What this book covers**

*Chapter 1, Setting Up Our First Project*, will discuss how to deal with Composer and Homestead. We will also cover the installation process of our very first Laravel project.

*Chapter 2, Building the Database with the Schema Builder Class*, will discuss the Schema Builder Class. We will analyze everything you can do with the class, look at different types of indexing, and learn about the methods that the Schema class provides.
Chapter 3, *The Most Important Element – the Model!*, will help us implement some "create," "read," "update," and "delete" logic for our items. We will also explore some useful methods and features of the model class.

Chapter 4, *Exploring the World of Relationships*, will help us discover how to work with different types of relationships and how to query and use them in a comfortable and clean way. Also, we will learn how to insert and delete related models in our database, or update existing ones.

Chapter 5, *Using Collections to Enhance Results*, will talk about collections. We will work with some results transformation methods and with the elements that make up a collection.

Chapter 6, *Everything under Control with Events and Observers*, will allow us to learn everything about the events in the context of Eloquent models. Right after, we will cover model events and model observers.

Chapter 7, *Eloquent... without Laravel!*, will explore the structure of the database package and see what is inside it. After that, we will learn how to install the "illuminate/database" package separately for your project and how to configure it for its first use. Yes, exactly: Eloquent without Laravel!

Chapter 8, *It's Not Enough! Extending Eloquent, Advanced Concepts*, will explore two different ways to extend Eloquent, and move on to learn about the Repository pattern.
Setting Up Our First Project

"Chi ben comincia è a metà dell'opera."

(Italian for "Well begun is half done.")

Every journey has a beginning, and only a hero with the right equipment can attain a victory. Of course, there are no exceptions for the 21st century hero: the developer!

In order to avoid problems and fight the bad (and malfunctioning) code monster, the good code artisan will prepare everything necessary before the start.

A developer has to be comfortable with the tools they are going to use, and a good development environment can awesomely improve the process. So, before getting our hands dirty, in this chapter, we will discover how to deal with Composer and Homestead.

Composer is an awesome dependency management tool, which is used by many PHP projects around the world. Homestead is the official Laravel Vagrant box that lets you create a fully functional development environment on a dedicated virtual machine in a matter of minutes. Finally, we will cover the installation process of our very first Laravel project.

I know what you are thinking: you just want to write code, code, and more code.

Be patient for a little while: if you know the tools we are going to analyze, at the end of this chapter, you will feel an enormous difference.
Trust me.

- Your Swiss Army Knife: Composer
- Your safe place: Homestead
- The new hideout: Homestead improved
- A bonus tool: Adminer
- Your best friend: Laravel
- Your first project: EloquentJourney
- Summary

Your Swiss Army Knife – Composer

The very first thing you will need to work with Laravel (and then Eloquent) is Composer. Composer is a dependency management tool for PHP. With this tool, you can easily include every dependency that is needed in your project. This is done in seconds, using a JSON configuration file named `composer.json`.

Usually, dependencies in a PHP project were managed with PEAR or other methods. Composer has a different policy: everything works on a per-project basis. This means that you can have two projects on the same server with different versions of the same dependency package.

Installing Composer

The installation procedure is ridiculously easy. All you have to do, is to go on the Download page of the Composer website and find the right method for your operating system.

- If you have Linux or Mac, just use this:
  
  `curl -sS https://getcomposer.org/installer | php`
  
  Or, if you don't have cURL, then use this:
  
  `php -r "readfile('https://getcomposer.org/installer');" | php`
  
  Also, the `composer.phar` file will be downloaded in your current directory.

- On Windows, you can simply download the dedicated installer.
Once Composer is installed, I suggest putting its path in the `PATH` variable of your system, in order to use it wherever you want. There are many ways to do it, which depend on your operating system. Let's look at each.

- On Linux, you can move Composer to the right directory simply with the following command:

  ```bash
  mv composer.phar /usr/local/bin/composer
  ```

- The same goes for OS X, but sometimes, the `usr` directory doesn't exist. You must create `usr/local/bin` manually.

- Finally, on Windows, you must open the control panel and type `environment variable` or something similar. The search utility will do the rest for you. Once in the right window, you will get a list of all environment variables. Find `PATH` and add the composer installation path to it.

### The `composer.json` and autload files

Before we go deep into our project, let's take a look at how Composer works.

In the `composer.json` file, the developer specifies every single dependency for its project. You can also create your packages, but we are not going to look at how to create them in this book.

So, let's say that you want to create a project that uses Monolog for logging purposes.

1. Create a folder for the project, then create an empty text file, and name it `composer.json`.
2. Open it and all you will have to do is to include your dependency as shown:

   ```json
   {
     "require": {
       "monolog/monolog": "1.12.0"
     }
   }
   ```

3. After that, save the file and type the following in your project directory:

   ```bash
   composer update
   ```

Wait a minute to download everything, and then you are done!

What? OK, here is how it works: Composer downloads every package you may need and automatically creates a loader for all your packages. So, to use your dependencies in your project, you will just need to include `vendor/autoload.php`, and you are good to go.
Setting Up Our First Project

Let's say that you have an `index.php` file as a start file for your application. You will have to perform something like the following:

```php
<?php // index.php file

    require('vendor/autoload.php');

    // your code here...
```

Nothing more!

Why am I showing this to you? Well, Laravel and Eloquent are Composer packages. So, in order to use it and create a Laravel application, you have to know how the mechanism works!

The most used commands

Composer is a command-line tool. Every good CLI tool has some important commands, and in this little section, I will show you what we are going to use the most.

- First of all, we have the following:
  ```bash
  composer create-project
  ```

  With this command, you can create a new project using a specific package as a base. You will use this command to create a new Laravel project using the following syntax:

  ```bash
  composer create-project laravel/laravel my_project_folder
  ```

- Then, you can find:
  ```bash
  composer install
  composer update
  ```

  These are two similar commands; they are similar, but not the same. When you specify your dependencies in the `composer.json` file, you can use `install` to install them. If you already installed them but you want to update your dependencies to a newer version, use `update`. 
In order to know what must and must not be updated, Composer uses the `composer.lock` file, which you can see in the root of your project. Actually, you will never have to work with it, but it's important to know that Composer uses it as a log of what it does.

- Sometimes, you will also see this:
  ```bash
composer require
  
  You can use `require` to include dependencies in your project on the fly. Here's an example of Monolog inclusion using `require`:
  ```
  composer.phar require monolog/monolog:1.12.0
  ```
- Another often used command is:
  ```bash
composer dump-autoload
  ```
  This command regenerates the `autoload.php` file. It can be useful if you add some classes into your projects without using namespaces or PSR conventions and rules.
- Sometimes, you will have to use (after a warning):
  ```bash
composer self-update
  ```
  This command updates Composer itself. Just a few seconds, and you are up and running again!
- Finally, you can use the following special command:
  ```bash
composer global COMMAND_HERE
  ```
  Use it to execute a specific command in the Composer home directory. As I mentioned before, Composer works on a per-project basis, but sometimes you will need to install some tools globally. With the global command, you can do it easily.

That's all you need to know about Composer right now, and yes, there are many other commands, but we don't need them now.

Let's take a step forward: it's time to learn about Homestead!
Your safe place – Homestead

When we start a new project, we might also stumble upon many compatibility and environment issues. The first one to think about is the PHP version. Maybe you are using XAMPP or some preconfigured stack on your local machine. For your new project, you want to use PHP 5.6, but the installed version is 5.3 (as you used it for some older projects). Fine, no problem; you can just install 5.6, and you are good to go.

Yes, but after two days, the phone rings. It’s your old customer; finally, it’s time to make some improvements and add new features! So, you start your stack services, browse your old project index, and BOOM! Compatibility issues, compatibility issues everywhere! Not exactly the best way to start your day.

This is not a code problem, but an environment problem.

Actually, the best solution is to start using Vagrant. Vagrant is a fantastic tool that lets you create a virtual machine with a headless operating system in order to configure the virtual machine on a per-project basis. Also, you can share some folders from your local machine with that machine, so you can work on an isolated environment while working with your favorite IDE and operating system.

Note that the per-project basis is the most important part of the entire thing. If you configure a separate machine for a single project, you can tweak everything you want to reach the perfect environment. Also, with Vagrant, you will be able to set your local environment in the same way your production machine is configured. So, no more local to production bugs and issues!

Last but not least, the fun (and useful) thing about Vagrant is that you can put a specific box under version control. So, for every new team member, all you have to do is to clone the repository and start the machine.

This looks complicated, but it is not. With Vagrant you can easily download a box (a ready-to-use virtual machine with all the tools and applications you need) and start it with a simple command from the shell as shown:

$ vagrant up
The Laravel community knows a couple of things about Vagrant and makes up a Vagrant Box to help you in your job.

```
$ homestead up
Bringing machine 'default' up with 'virtualbox' provider...
  --> default: Checking if box 'laravel/homestead' is up to date...
  --> default: Clearing any previously set forwarded ports...
  --> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
  --> default: Forwarding ports...
    default: 8000 => 8000 (adapter 1)
    default: 443 => 44300 (adapter 1)
    default: 3306 => 33060 (adapter 1)
    default: 5432 => 54320 (adapter 1)
    default: 22 => 2222 (adapter 1)
  --> default: Running 'pre-boot' VM customizations...
  --> default: Booting VM...
    --> default: Waiting for machine to boot. This may take a few minutes...
      default: SSH address: 127.0.0.1:2222
      default: SSH username: vagrant
      default: SSH auth method: private key
    --> default: Warning: Connection timeout. Retrying...
      default: Warning: Remote connection disconnect. Retrying...
      default: Warning: Remote connection disconnect. Retrying...
    --> default: Machine booted and ready!
    --> default: Checking for guest additions in VM...
    --> default: Setting hostname...
    --> default: Configuring and enabling network interfaces...
    --> default: Mounting shared folders...
      default: /vagrant => C:/Users/Francesco/AppData/Roaming/Composer/vendor/laravel/homestead
      default: /home/vagrant/Code => C:/Users/Francesco/Code
    --> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
  ==> default: to force provisioning. Provisioners marked to run always will still run.
```

Homestead is the official Vagrant Box for Laravel and already has everything you need to get started. You will find it (already installed and working), by default:

<table>
<thead>
<tr>
<th>Ubuntu 14.04</th>
<th>Node (with Bower, Grunt, and Gulp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP 5.6</td>
<td>Redis</td>
</tr>
<tr>
<td>HHVM</td>
<td>Memcached</td>
</tr>
<tr>
<td>nginx</td>
<td>Beanstalkd</td>
</tr>
<tr>
<td>MySQL</td>
<td>Laravel Envoy</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Fabric + Hipchat Extension</td>
</tr>
</tbody>
</table>

Not too bad for a tool box that you can prepare in a matter of minutes!

Now let's stop discussing and install Homestead.
Installing Homestead
First of all, ensure that you have already installed VirtualBox (https://www.virtualbox.org/) and Vagrant (https://www.vagrantup.com/). You can install them on every operating system, so feel free to use whichever you want.

If you want to work with a good shell on Windows, I suggest you use Cmdr (http://bliker.github.io/cmder/). While writing this book, I referred to the same link.

Next, we can add Homestead to our local boxes. This means that Vagrant will download the Homestead box in order to be used locally.

You can do it with a simple command:

```bash
vagrant box add laravel/homestead
```

You will have to wait a couple of minutes to download the box. So, if you want to have a coffee, this is the perfect moment.

Here, you don't have to worry about where Vagrant is placing the box, as it is going to save it locally in the Vagrant folder. In the future, every time you will need a specific box, Vagrant will clone and use it.

Alright, your box is now on your local machine and ready to be started. However, accordingly to your local machine settings, you can install Homestead in two different ways. They are both present in the official Laravel documentation, so they are both official.

The Composer and PHP tool way
Let's start with the first one: it is a perfect choice if you already have Composer and PHP on your local machine. Note that you are only going to do these steps the first time.

Use this command to install the Homestead CLI tool.

```bash
composer global require "laravel/homestead=2.0"
```

Then, be sure to put the ~/.composer/vendor/bin directory in the PATH environment variable, in order to use the tool wherever you want.
After that, you can initialize your machine. Use the `init` command:

```
homestead init
```

This will create a `~/.homestead` folder with a `Homestead.yaml` inside of it. This file will be used by Vagrant at the virtual machine start.

### The Git way

If you don't have PHP and Composer installed on your local machine (or maybe you just don't want to use them), no problem. You can simply use Git.

Choose a folder where you want to save your virtual machine. Then, clone the repository with:

```
git clone https://github.com/laravel/homestead.git HomesteadFolder
```

Here, `HomesteadFolder` is the place you chose for your VM files. After the clone process, use `cd` to get into the folder and start the `init` script using the following command:

```
bash init.sh
```

This script will create a `Homestead.yaml` file in a `~/.homestead` directory, and that's it!

The following steps for installation are the same for both the methods you just saw.

### Configuring Homestead

Before we go forward, let's take a look at the default `Homestead.yaml` file.

```yaml
---
  ip: "192.168.10.10"
  memory: 2048
  cpus: 1
```
Setting Up Our First Project

authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
    - map: ~/Code
to: /home/vagrant/Code

sites:
    - map: homestead.app
to: /home/vagrant/Code/Laravel/public

databases:
    - homestead

variables:
    - key: APP_ENV
      value: local

If you are unfamiliar with this syntax, no problem; it's a simple YAML (YAML ain't a markup language) markup file. It is a very readable way to specify settings, and Homestead uses it. Here, you can choose the IP address for your virtual machine and other settings. Tweak the configuration file accordingly to your needs.

1. Do you see the authorize property in the Homestead.yaml file? Well, we are going to set up our SSH key and put its path there. If it scares you, don't worry; it is just a command.

```
ssh-keygen -t rsa -C "you@homestead"
```

If you are using Windows, the Laravel documentation recommends Git Bash. Personally, as I mentioned before, I prefer to use Cmder. However, you can also use PuTTY or whatever you want. Use `ssh-keygen -t rsa -C "you@homestead"` to generate your ssh key. This is shown in the following screenshot:
2. Put the generated SSH key path in the `authorize` property of Homestead. `yaml`, as shown in the following:

```yaml
authorize: ~/.ssh/id_rsa.pub
```

Done? Good. Now, you can see a `folders` property as well.

As I mentioned before, Vagrant lets the developer share some folders between the local and virtual machines. What is the point of that? Well, it is really important because with this system we can work on our project on a separate machine, while being able to use whatever IDE or tool we want from our local machine. For example, even if the VM has Ubuntu, I can easily use Windows 8.1 and PHPStorm. The best of both worlds!

3. By default, Homestead suggests this structure:

```yaml
folders:
  - map: ~/Code
to: /home/vagrant/Code
```

Also, this means that you will have to create a `Code` folder in your user folder. This local folder will be mapped to a `/home/vagrant/Code` folder on the VM; every change that you make there will be reflected on the virtual machine and vice versa.

You can customize this mapping to your needs.

4. Next, let's take a look at the `sites` property. Here's what you can see in a default setup:

```yaml
sites:
  - map: homestead.app
to: /home/vagrant/Code/Laravel/public
```

You can define a custom domain for every project, which is a really comfortable way to work with your projects, as you will no longer need to test your project with an IP (like 192.168.10.10), only a simple local domain, such as `myproject.dev`. 
5. This is a good point to define a separate site for our project. So, feel free to add these lines to your file:

```yaml
- map: eloquent.dev
to: /home/vagrant/Code/EloquentJourney/public
```

6. Next, go to your host's file (on the host machine) and add this record:

```
192.168.10.10 eloquent.dev
```

You can see how you need to add it in the following screenshot:

```
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a `#' symbol.
#
# For example:
#
#   102.54.34.97   rhino.acme.com # source server
#   38.25.63.10   x.acme.com     # x client host
#
# localhost name resolution is handled within DNS itself.
# 127.0.0.1      localhost
# ::1            localhost
192.168.10.15 eloquent.dev
```

Of course, you have to insert the same IP you specified in the Homestead.yaml file.

7. The last thing we are going to see here is the database property. For every name you add here, Homestead will automatically create a database to work with. So, edit the property to something like this:

```yaml
databases:
- homestead
- eloquent_journey
```

This is because we are going to use a separate eloquent_journey MySQL database for our test application.
Chapter 1

The default username for the MySQL server is homestead, and the default password is secret.

We have nothing more to do here; our setup is complete, and now we are ready to boot up our virtual machine and use it.

The new hideout: Homestead improved

Even if Homestead is a fantastic box, many people complain about some of its structural choices. As I mentioned previously, Vagrant is used to create virtual machines on a per-project basis. This means that, in an ideal situation, every project must have its own VM. Now, with Homestead, you can create a single VM and manage all your projects on it. Some people like this idea, and it is more familiar to the classical XAMPP approach. Quite familiar!

However, other people like a more pure approach to Vagrant. While doing some researches on this concept, I stumbled upon Homestead Improved (https://github.com/Swader/homestead_improved) by Swader, on GitHub.

It is an improved version of Homestead that you can install and run without saving files all around your user folder. A really good approach! Also, you won't have to configure any SSH keys or execute `apt-get update` and `composer auto-update`. Everything will be done automatically.

If you want to use Homestead Improved, just open your terminal and type the following:

```
    git clone https://github.com/Swader/homestead_improved.git
    MyHomesteadImprovedVM
```

Here, `MyHomesteadImprovedVM` will be the containing folder of all the VM's files.

After the clone procedure, just type the following:

    vagrant up

So, you're done! Easier than before, isn't it?
A bonus tool – Adminer

Before going deeper along our journey, there is another really useful tool that I want to show you. I am talking about Adminer, a Database Management tool entirely contained in a single .php file. You can download it at http://www.adminer.org/.

Maybe you will find the Adminer interface very similar to the phpMyAdmin interface. It's true, but Adminer has more features. Just to make a simple example, phpMyAdmin only supports MySQL. Instead, Adminer supports MySQL, PostgreSQL, SQLite, Oracle, and MS SQL.

![Adminer Interface]

Obviously, you can use whatever you want to deal with your database. However, I wanted to show you Adminer because it is what I am going to use to show, from time to time, some query results or various examples. So, it would be good if you get more familiar with this tool.

Your best friend: Laravel

We are close to the end. You have a weapon (Composer) and a safe place to do everything you want without worrying about issues (Homestead). What about an ally? Laravel could be a good one, don't you think? Also, Laravel is the Eloquent container: we are going to create a new project with it to fully embrace its power.
Installing Laravel

Before going further, remember that Laravel has some prerequisites. You will need the following:

- PHP 5.4 (or more recent versions)
- The PHP Mcrypt extension
- The PHP OpenSSL extension
- The PHP Mbstring extension

If you are using PHP 5.5, you may need to install the JSON PHP extension. If this is the case, just type this:

```bash
apt-get install php5-json
```

So, you are good to go.

Obviously, if you have installed Homestead, everything is already in its right place.

1. All you have to do is to boot up the VM with the following command:
   ```bash
   homestead up
   ```

2. And when the bootstrap procedure is done, use the following command to get in the machine via SSH:
   ```bash
   homestead ssh
   ```

Having said that, as you may have experienced from Homestead, Laravel also gives you two different ways to install it and create a new project.

- The first one is done using a specific tool, the Laravel installer tool. It’s a CLI tool that you can install as a global Composer package.
- The second one is a simple `composer create-project` command. Of course, we will now see both ways.

### Using the Laravel installer tool

The Laravel installer tool is a nice utility that lets you create a new Laravel project with a very simple syntax. Imagine that you want to create a new project in a folder called `my_project`. All you have to do, if you have the tool installed, is to type this and nothing more:

```bash
laravel new my_project
```
Setting Up Our First Project

Installing the tool is easy. Just open the terminal and type the following:

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

As you saw before, we are executing the `require` command with the `global` keyword. This means that the installer tool package will be saved in the Composer's `global` folder and the tool will be available everywhere.

If you have any problems running the tool, just be sure to put 

```bash
~/.composer/vendor/bin
```

in the `PATH` environment variable. Otherwise, it won't work!

Using the Composer `create-project` command

If you don't want to install the Laravel installer tool, you can simply use the `create-project` command of Composer.

All you have to do, in this case, is use this command:

```bash
composer create-project laravel/laravel ProjectName
```

Here, `ProjectName` stands for the folder name that you want to use as the root of your new Laravel project.

Nothing more to do here! Your Laravel project is now fully installed in your specified folder.

Be sure to configure the right permissions on your folders and ensure you take a good look at the URL rewriting rules. If you take a look at the Laravel-dedicated documentation page (http://laravel.com/docs/5.0/installation#pretty-urls), you can learn how to do it on Apache or nginx.

The first project – EloquentJourney

A new project will be the perfect metaphor for our new, fantastic journey! While studying Eloquent, we will build a simple project. More specifically, we will analyze a hypothetical library management system's data-related part and its components.
What are you waiting for? Let's start! First of all, create a new project (using your favorite method). We will call our new project **EloquentJourney**. Type the following in your server folder:

```
laravel new EloquentJourney
```

Otherwise, type the following if you prefer:

```
composer create-project laravel/laravel EloquentJourney
```

Wait a few seconds to build the project, and after the installation procedure, you are done! You can use `cd` to get into your new folder and see what's there.

Cool! All right, but what are we going to do now? There are thousands of files here and in other subfolders! Don't worry. Take a breath and follow me. First of all, we need to do some practice with the Laravel configuration system in order to set up an appropriate database connection.

Without it, we could not use Eloquent!

## The configuration system

Everything you could need on configuration is stored in the `config` directory. Every file here has quite a descriptive name: `app.php`, `database.php`, `filesystems.php`, `cache.php`, and so on. Actually, we are going to use two of these files: `app.php`, for some basic settings, and `database.php`, for obvious reasons.

First of all, let's open the `app.php` file and see what you can find inside.

```php
<?php

return [
    'debug' => true,
    'url' => 'http://localhost',
    'timezone' => 'Europe/Rome',
    'locale' => 'en',
    'fallback_locale' => 'en',
]
```
Setting Up Our First Project

A Laravel config file contains a return instruction. The returned value is an associative array. As you can easily imagine, the key-value system represents the configuration item name and its value. For example, let's examine the first item:

'debug' => true,

This means that the app.debug configuration item is set on the Boolean value true.

Laravel uses these values all around the framework code, and you can use them too with the \Config class.

Specifically, if you want to retrieve a specific item value, you have to call the get() method, as follows:

```php
$myItem = \Config::get('item.name');
var_dump($myItem);

// true
```

You can also set a specific Config value at runtime, this time using the set() method, as follows:

```php
\Config::set('item.name', 'my value!');
$myItem = \Config::get('item.name');
var_dump($myItem);

// "my value!"
```
Setting up the database connection

Yes, we finally arrived at the end of this chapter. The last thing we need to do here, is to set up the database connection.

Let's open the database.php file under config. You should see something like this:

```php
return [
    'fetch' => PDO::FETCH_CLASS,
    'default' => 'mysql',
    'connections' => [
        'sqlite' => [
            'driver' => 'sqlite',
            'database' => storage_path().'/database.sqlite',
            'prefix' => '',
        ],
        'mysql' => [
            'driver' => 'mysql',
            'host' => 'localhost',
            'database' => 'homestead',
            'username' => 'homestead',
            'password' => 'secret',
            'charset' => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix' => '',
            'strict' => false,
        ],
        'pgsql' => [
            'driver' => 'pgsql',
            'host' => env('DB_HOST', 'localhost'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'charset' => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix' => '',
            'strict' => false,
        ],
    ],
];
```
Setting Up Our First Project

```php

 pouch

'prefix' => '',
'schema' => 'public',
],

'sqlsrv' => [
'driver' => 'sqlsrv',
'host' => env('DB_HOST', 'localhost'),
'database' => env('DB_DATABASE', 'forge'),
'username' => env('DB_USERNAME', 'forge'),
'password' => env('DB_PASSWORD', ''),
'prefix' => '',
],

],

'migrations' => 'migrations',

'redis' => [
'cluster' => false,

'default' => [
'host' => '127.0.0.1',
'port' => 6379,
'database' => 0,
],

],

];

The two most important items are default and connections. In this second item, connections, we are storing all the information we need to connect to our databases. By default, you will find many examples. In fact, here you can see the sqlite, then mysql, and also sqlsrv connections.

Every connection has a driver. The driver element indicates the used database for that connection. If you want, you can specify more than one connection, when necessary. The default element represents the chosen connection.

---

[20]
Let’s delete everything and replace the default and connections elements with the following:

`'default' => 'eloquentJourney',

'connections' => [

    'eloquentJourney' => [
        'driver' => 'mysql',
        'host' => 'localhost',
        'database' => 'eloquent_journey',
        'username' => 'homestead',
        'password' => 'secret',
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
    ],
],

What did we just do?

Quite simple! We have defined an eloquentJourney connection. This connection will use the mysql driver. So, we are going to connect Laravel to a MySQL server. I am not going to explain the other properties, as it is really easy to understand their meanings.

After that, we specified the connection name as the default option. This means that, for every future call to a database-related operation, Laravel will connect to the server specified in the eloquentJourney connection with the given credentials.

**Summary**

We did it!

We prepared everything that is needed to work with Laravel and Eloquent. We set up a local development server, learned the basics of Composer to correctly manage our dependencies, installed a couple of more useful tools, and, finally, successfully configured our database connection. Not bad for the first chapter, huh?

However, we have just begun, and our journey inside Eloquent is at the very beginning. We are ready to leave our safe house, go into the darkest corner of the Eloquent ORM to explore it, and understand all of its secrets.

It will be a great ride. And now, let’s explore our first topic on our way: the Schema Builder and the migrations system, to build the perfect database!
Where to buy this book

You can buy Learning Laravel's Eloquent from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.