crafting

Welcome, I'm Dieter, i'm a web application developer.
I'm building software using the Laravel ecosystem and want to give you my some tips and tricks. I hope you will learn some cool stuff !

Dieter Coopman
co-owner and developer, Delta Solutions

Table of Contents

About me

HI there, I’m Dieter

Hello I’m Dieter Coopman, developer and co-founder of **Delta Solutions**, located in Waregem, Belgium, founded in 2002. We build custom-made software solutions with the Laravel stack.

I’m programming in PHP since then, and I’m an **early adaptor of the Laravel framework**. Working with Laravel started for me at version 4.0 in 2013, and I’m still a huge fan. Frontend work I do, nowadays, is mainly in Vue.js combined with Bootstrap, Vuetify, Element UI. I Also crafted some pieces in software in Laravel Livewire.

Learning new stuff

I **love experimenting** and exploring new things, trying to extract the good things that can be useful for our team at work !

With this writing **I want to contribute in the learning and exploring process** of developers and give my opinion on things I’ve learned.

In my spare time I love enjoying my wife and 2 kids, my family, running, cooking and off course... Programming and exploring new stuff

You can follow me or **get in touch on social media via my social feeds** my twitter handle is @dietsedev. This is my Github page. You can email me via hello@dietse.dev

This is my setup

My computer

Hi, curious about my setup ? I'll tell you what I use …

I'm working on a **13″ Macbook Pro M1**. It has **16GB** of RAM and **512GB** of disk storage.

I'm using a wireless **Apple keyboard**, the best keyboard I've ever had. The mouse I'm using is an **Apple Magic mouse 2**, yes that with the cable connection on the downside (impossible to charge when using)

I'm using an **Iiyama 27″ external display** with a resolution of 2560×1440

My dev tools

PhpStorm is my favourite IDE, i've tried VSCode a couple of days but took back to the IDE i'm used to work with.

I’m using **Querious** as SQL editor for MySql, its a fluent working tool with a lot of possibilities.

iTerm2 is my favourite terminal application, extended with Oh my zsh

Tower is the git client I’m using, it’s a very nice application if you prefer (like me) a visual way of managing your git repo’s. I’m using it most of the time with **Gitlab** / **Bitbucket**.

Transmit is my favourite ftp tool, one of the benefits of it is that you can manage two ftp locations side by side and connect with S3 compatible storage.

Other productivity tools

I use **Alfred** to automate some tasks like opening my dev tools or search through the Laravel documentation. I know it more than ten years but I have to admin that I never explored its possibilities through the powerpacks.

I’m reading my mails with **Outlook**, it gained a lot of power and possibilities the last few years and is included in the MS Office suite.

My R&D and test environment is a ~~Docker setup~~ **Valet setup**, very handy if you want to test things.

Now I'm using Valet because of the lack of multiple hosts in Sail. But working with Sail feels very, very nice. You install a docker environment, test your application, and if your work is done you throw it away. Without traces and installations on your local machine.

I manage my open windows via **spectacle**, with this little tool it is very easy to pop a screen to the left or the right of your monitor.

What are you using

You have nice productivity tools that you want to share with me ? Get in touch via social media !

This is the story of how I became a developer !

What is this article about ?

I’ll give you an overview of my evolution as developer. How and when I started programming and what journey I made through the past 20 years.

Education

In highschool I studied economics, I wasn't really meant to be a software developer at that time. After highschool I didn't really know what to study, and I followed a friend. Starting a multimedia education, became a bachelor in multimedia three years later in 2002. My friend stopped the courses after 3 months, I went through.

Programming

I never programmed before the age of 19, at that time it was not so common practice learning programming at highschool.

In the three years of my education I’ve learned a lot of programming languages like Java, Visual basic, C … We learned Linux, Macromedia Flash, some multimedia tools, photography, … Educated to become multimedia developers, with a very broad base … We learned things that informatics students didn't learn at that time and were “the new kids in town”, multimedia was hot, we were hot !

Web application, what you say ?

At that time there was no big focus on web languages. But we learned Flash, Actionscript, HTML, CSS, Javascript and ASP … The before .Net era. I think (am not sure) we haven’t seen PHP at school, and I made my graduation work in ASP with an MySql database.

Programming was done in notepad. I did my graduation work together with the guy I will later start my company.

My work

In 2002, on August 19 (the day of birth of my sister) I started Delta Solutions, with Steven Pauwels as co-founder. We were young, only 21 years old and could kickstart our career via the connections of Danny (the person we made our graduation work for – still many thx for that push). He motivated us to become self-employed, and we rented an office from a company he knew. That company tried to pick up with web technology.

New kids :-)

They were using weird unknown technologies, in fact desktop software that made a web toolkit. The young guys (we) knew new stuff, real web stuff and so the story started … We picked up the web work. A few years later our ways separated and Delta Solutions upped the pace exploring new stuff and new work . Many thanks to Koen and Bart for the 5 years of intensive cooperation.

The steel climb of the web

That time, in 2002 it was really hard to convince companies to work with web technology. Many companies didn't see the potential of the web. At that time everyone wanted desktop apps (now knowing better 20 years later).

Technology

We started Delta Solutions with a boxed version of a program called Dreamweaver, an Adobe tool that could handle php. That time we didn't talk about mvc, oop. We had one php page that handled everything, from logic to database communication to view processing. We were using plain javascript with iframes to prevent page refreshes, in the pre-ajax era. I was using prototypejs in 2005 before jQuery was created in 2006. I picked up with jQuery and left prototypejs for what it was. At that time I didn't use any CSS frameworks (bootstrap was created in 2010). We crafted all the styling and interactive stuff ourselves.

Evolving from Dreamweaver functional programming to pre-framework oop with own created classes to make things easier we were using Netbeans, that was in fact a Java environment. We picked up the new things after testing them, early adaptor of the things that gave our development steroids.

Nowadays we still use the top notch technologies, stuff like Vue.js, Livewire, Ionic ... still in front of the line when it comes to new stuff.

Frameworks

Around 2010 the first php frameworks (symphony, codeigniter) were growing interest, and we decided to give Codeigniter a try. We made a project in it, but saw at the same time that the creator of the framework was seeking a new owner for CodeIgniter, citing a lack of resources to give the framework the attention they felt it deserved. Taylor must have read this :-)

We created one project in Codeigniter, around that time we read about version 4 of Laravel and made an u-turn picking up Laravel for our next project … One of the best decisions we have ever made. It’s the summer of 2013 at that time. It was the start of real php ecosystems and tools like composer, that was another boost for our toolset and possibilities.

Laravel

Still a daily user of Laravel, and it’s ecosystem. Evolved from crafter with own created libraries to community driven development using tools adapted worldwide, tools like gulp, webpack, composer, bootstrap, vueJs, elementui and many, many more.

7 years later I still try to use top-notch technologies ? I try to pick up the new stuff that fits best. I have to admit that it became harder and harder to get the right stuff, webtechnologie and toolsets are booming, and it’s harder than ever to pick the right one … Keeping it very interesting to do my job. I love it more than ever. Hope you also do !

Embrace the commandline

Introduction

If you want it or not, the commandline is part of your daily toolset as developer. As a Laravel developer you will use `artisan`, as a javascript developer you will use `npm`. As a web developer you will use the commandline on a daily basis. We have to embrace the commandline, and once you do so, it will give you extra powers.

What is the power in it ?

If you embrace the commandline then you will see that a lot can be done faster. By creating aliases to commands you can even make the commands shorter. An alias can be created by typing `alias [youralias]='[your command]')` , for example `alias pam='php artisan migrate'` will run the php artisan migrate command when you type `pam`.

I've created aliases to directories on my filesystem, so I can jump to them very fast. Typing `open .` opens finder on the location I jumped into. `pstorm .` opens phpstorm for the given directory and `gittower .` opens git tower for the directory. If you now combine this in an oneliner and create an alias for all this together you can get this ...

```
alias work='open . && pstorm . && gitower .'
```

If you now type `work` in a directory your productivity tools will open. How cool is that!

Some commands I use almost daily

The commands like `ls` , `mkdir` , `cd` are known by almost everyone, but there are a lot of handy commands that are less well-known but are nevertheless very useful. I'll try to give you some commands that I use **A LOT** and that might help you to speed up your workflow. I've already spoiled a couple commands in the previous chapter.

alias

It can be very handy to create an alias for common used commands. These aliases are shortcuts to a given command. Aliases are saved in your user profile and will be remembered by your system.

An alias can be created by typing `alias [youralias]='[your command]')` , for example `alias pam='php artisan migrate'` will run the php artisan migrate command when you type `pam`.

&&

With `&&` you can combine multiple commands in one line, this is very handy in combination with an alias. You can create aliases that do multiple things when calling the alias.

tail

Do you want to see new entries written into some log file, maybe the laravel.log file of your application. Then tail is your companion command. If you tail a file, the last lines of the file will be shown. By default, tail prints the last 10 lines of each file to standard output. If you specify more than one file, each set of output is prefixed with a header showing the file name.

If you want to output more than the default of 10 lines you can use the `--lines=num` or `-n num` option, specifying the number of lines you want to show.

It is also possible to let tail output any new line added to the file your are looking into. So if a new log entry is written to the file, it will immediately be shown in your output. This can be done via `--follow` or `-f` as option.

history

If you type `history` you will get a list of the n last commands that you did run on your computer. This gives you the possibility to copy past commands that you executed in the past.

As mentioned by [Jason Judge] in a reply on twitter you can also use control-R to search through your history to go back to a previous command.

|

the `|` , the pipe command makes it possible to chain commands. So you can use the output of a previous command to a next command. This is very handy in combination with `grep`. You can let a command output something to your screen, pipe it to another command, so it will not be shown on your screen but passed to the next command.
In combination with `grep` you can now search in the output.

grep

Grep gives you the power to search into files. `grep -Ril "string to search" *` Will search all files in the current directory and show you the filename of the file where the string is found in.

`i` stands for ignore case (optional in your case). `R` stands for recursive. `l` stands for "show the file name, not the result itself".

Grep can also be used to search in piped output, I use the combination of `history` and `| grep` *a lot* . For example , if you have multiple of versions of php installed, and you don't actually can't get the directory top of mind where php8 lives, but you know you ran a command with php8 last week then you can search your history with the following command.

```
history grep |  php8
```

chown

`chown` makes it possible to change the ownership of a file. Via these commands we will change the owner of the files to 'www-data' and assign the files and folders to the 'www-data' user group. Making it possible for apache to write into this directories.

```
chown -R www-data:www-data *
```

curl

You might know the term curl-request, but did you know that `curl` is a terminal command. You can use it to test get requests on the commandline.

touch

The `touch` command allows you to create a blank new file through the Linux command line. As an example, enter `touch /home/username/Documents/Web.html` to create an HTML file entitled Web under the Documents directory.

pwd

The `pwd` command shows the path you are currently working in.

df

Use `df` command to get a report on the system's disk space usage, shown in percentage and KBs. If you want to see the report in megabytes, type `df -m`.

du

If you want to check how much space a file or a directory takes, the du (Disk Usage) command is the answer. However, the disk usage summary will show disk block numbers instead of the usual size format. If you want to see it in bytes, kilobytes, and megabytes, add the `-h` argument to the command line.

Retrieving the size of all files and folders in a given directory can be done via following command `du –sh`.

If you want to receive the total size of a given folder including the sum of all subfolders you can do this via this command `du -h --max-depth=1`.

ps

`ps` displays information about a selection of the active processes. If you want a repetitive update of the selection and the displayed information, use top instead. Ps will show a list of processes , including a process id. I you want to terminate a process you can use `kill pid` to terminate the process.

Sajan

Because there are many commands that are hard to remember, or combinations of commands that we often run together I've create sajan. It's a commandline tool that gives you some shortcuts to this difficult to remember commands. You can find more information about sajan on Github.

Learn more

I've contributed to a Linux book. It explains the most popular Linux commands, it's a completely open source driven book and can be downloaded here. If you also want to contribute to the book, you can take a look into the GitHub repo

I'm in love with Laravel

I really love Laravel

Are you asking **why I love Laravel** ? Ok do you have a day or two ?

Laravel is **one of the best things that crossed my path the past 20 years.** It's a Php framework initially created by Taylor Otwell with a great goal in mind. **Keep it simple and clear.** This goal is achieved by wrapping all the heavy stuff in nice api's , making it **easy for developers** to do very complex things.

Documentation

First I love Laravel because of its documentation. It is the best documented tool I have ever used, and I know that Taylor Otwell, the creator of the framework, and his team are perfectionists in documenting ?

Features

Laravel is an easy to learn framework and has very powerful features implemented in a very clear way. Laravel aims to take away the really tough work for the developers by wrapping the complex setup in easy to use Laravel features.

Routing, eloquent, blade, collections … Everything rocks, and it is clear and simple to use.

Structure

It ‘forces’ you to work structured and in a recognizable way. So it’s easier to pickup with someone else’s job. This is typical for frameworks, but I’ve started programming in PHP way before frameworks were born. Everybody created his own classes and tools at that time.

Extensibility

It is very extensible. If you need a job to be done, there will be a package to help you. Creating Excel exports, resizing images, crafting pdf’s, this can all be done by installing extra composer packages into Laravel.

The community

It has a huge community, so if something is not clear, ask it to the community. There are tons of video tutorials ? available and there are many code examples available all over the internet. There are reddit channels and discord channels available and core members of the Laravel are sometimes answering themselves.

Very expressive API

The code itself is also very **well-thought-out** and has a very **expressive API**. If you look at the pull requests of Laravel you can see that de developers reviewing the code are perfectionists and discussions about the code makes that only the best possible solution is added to the framework.

A clear vision

The vision of its community is very clear, and that's making the life of developers easier. Many community driven pieces of software are created by developers all over the world, some of them solving very complex problems with a very simple API. Simplicity as main goal, that rocks ▯.

The error handling

It has a gorgeous error page. It’s not only an error page but a very handy tool that specifies where your error is, and you can open your ide on the line of your error by clicking a pencil.

That’s why I love it !

Laravel Livewire vs Vue.js

My opinion on Livewire vs Vue.js

I’m using Laravel since version 4.0, and I’m still a big (in fact huge …) fan of it. When frontend (javascript) frameworks like Angular and React became popular I’ve noticed the high interactivity on the frontend that came with it … No need to refresh browser screens to retrieve new data and super reactive applications was the result of those frameworks. Updating stuff on your screen without complex javascript logic was (and still is) mind-blowing ! The frameworks took the web-development scene by storm ! At that moment the Laravel framework didn't really incorporate one of those frameworks. Until Vue.js was released somewhat seven years ago (late 2014).

Laravel decided to optionally include Vue.js in its Framework installation. So it was a logical decision to choose Vue.js as frontend framework, and so did I. Since 2016 I’m using Vue.js for all the new applications I build.

My opinion about Vue.js

In fact, I’m a fan of Vue.js. It gives the user a high interactive experience, and user is experience is the number one concern when building applications. Since I’m using Vue.js I never used jQuery again, and I write less javascript for building interactive stuff. So that’s a huge win. Better ui’s with less code to maintain. That's a win-win situation.

But ! While being a huge fan for interactivity I hate the bootstrap process. I hate the actions / mutations / getters (the vuex approach for handling data in bigger applications), bah … I hate the router, the slow build processes and the complexity when doing a lot of data handling in the frontend before we send it back to the server. Have you ever debugged javascript … That’s one of the biggest nightmares for me, no stack traces, no “error at line x” … But hey ! We have interactivity, the user will love our application that never refreshes. But, I am forgiving, so I still love it, and there is (was) no alternative that integrates that good with Laravel. And hey, everyone is using frontend Frameworks, so now I’m fancy too ? (same quote as Caleb in his video mentioned below)

And then I saw Livewire ?

One day I wanted to build a documentation platform for our company. When building stuff like that, from scratch, I love to try the new things that come along. I noticed Laravel Jetstream, a new tool, being a hobby project of Taylor Otwell. I could choose between a Vue.js + inertia implementation or the Livewire implementation. The what, Livewire … I never heard about it, and it caught my attention. I wanted to discover this new piece of software.

I discovered the playground of Caleb Porzio, a former Laravel / Vue.js developer at Tighten. I read all his stories, and I was an instant fan of that guy, and his tool(s) … That man describes the same feeling as I feel in this video – you really MUST see this video – somewhat te same as my chapter about Vue.js. Damn, why haven't I used Livewire before … oh yeah … the 1.0 release was in February 2020, it's brand new !!

The power of Livewire

What is Livewire ? Livewire is a full-stack framework for Laravel. You can only use it with Laravel because it is tightly linked to the backend. So it is impossible to use it for applications that live totally separate from the backend – it is connected to the backend. So that is the negative part … For those applications where separation of backend and frontend is a must, via an API approach, you'll have to stick to Vue.js.

Now the positive stuff ! Ready !?…

I'm using Livewire for a while now, testing its boundaries before using it in client projects. I'm testing it out with a friend of mine (mister V.), in a skeptic way. A skeptic way because we want to be sure that everything we do with Vue.js can be done with Livewire. Yesterday we (he in fact) tackled the last big challenge … Infinite scroll seemed a conceptual problem, but it is solvable in a neat and reusable way. Mister V. is even more skeptic than I am, but his enthusiasm is growing and the skeptic is disappearing !!

What is its power ?

The power of Livewire comes with the fact that it's all around pure Laravel. Your views will all be blade views, coupled to your backend via magic called 'wire'. You can wire the click action of a button directly to backend methods via a wire:click call. If you make a property in your backend publicly accessible then it will interact with your frontend via wire:model. This all without writing any javascript. The data handling can all be done in your backend, and you don't need to use stuff like Vuex and Axios to communicatie with your backend.

No build process

There is no build process. I think the development process can be speed up with 50%, while keeping interactivity in place for the end user.

You can still use all the Laravel superpowers like routing, validation, error bags and so on. And because of the interactivity added via Livewire your error messages will automatically become visible without refreshing your screen.

Is Laravel Livewire the next big thing ?

Really ?

It might be … If you **don't need a single page application** and if you don't freak out by the idea that your backend is not decoupled from your frontend.

It's a competitor for Vue.js, and now you can say "why did you start using Vue.js and not Livewire". Well euh … **It's brand new**, released in februari 2020.

What is Livewire, and what are the advantages ?

Livewire describes itself as follows

Livewire is a full-stack framework for Laravel that makes building dynamic interfaces simple, without leaving the comfort of Laravel.

Yes, you've read it well. You can create reactive applications without leaving your Laravel. In my opinion Caleb Porzio has done a massive good job by creating Livewire, and I'll tell you why.

Simplicity

When you are a Laravel developer you are used to working with blade templates. With Livewire this is also true. You can stay in the environment and syntaxis you are familiar with.

it's PHP

You even stay in PHP, in the simplest form you don't even need to write javascript.

Installation

Installing Livewire is as simple as "composer install". The package is installed into your application and ready to use. The package has some built-in artisan commands to speed up your development. No specific bootstrapping needed, it's just Laravel.

In a popular video on my Youtube channel I explain to you how the setup Livewire from scratch in less than 5 minutes. The code of the demo can be found here : https://github.com/dietercoopman/livewire-crud-demo

No build process

No builds needed ! In contrast to frameworks like Vue.js, you don't have to build your application. This is much faster than the npm build process that's needed to build your Vue.js application.

How do you use it ?

The full documentation can be found on the Laravel Livewire website but I give you a quick shot on how it works. In fact, if you are familiar with Vue.js or any other javascript framework, some terminology will feel very familiar.

Properties : all of your public properties defined in your Livewire component are accessible in your views.

Data binding : `<input wire:model="message" type="text">` This will bind the input model to the public message property of your component. This means that if you change your input, the component (backend) will get notified and the value will be changed.

Actions: `<button wire:click="like">Like Post</button>` If you click the button the like method will be called in your Livewire component.

Events : events can be fired and listened for via emit

How did I catch up with Livewire ?

I've noticed that Taylor created Jetstream with two hotly debated stacks. First stack is Vue.js + Inertia and the second stack is Livewire + blade. So this is where I picked it up, by reading documentation.

I say hotly debated because a lot of people found it irresponsible from Taylor to use such a new frameworks to build something like Jetstream. The answer from Taylor was, "hey can I please do what I want in my spare time … programming is my hobby, and I use what I like in my spare time"

The full answer of Taylor can be seen on his Youtube channel.

I'm doing open source for one year now

My goal ?

My 40th birthday was my trigger to look back. Looking back to what got me to where I am now. A web developer, co-owner of a company with a team of 7 people using open source software every single day. I wanted to give something back for all the great open source stuff we use every day.

So my idea was (and still is) to try to do some things

- I wanted to create at least one open source package
- I wanted to create a website with valuable information
- I wanted to share some information on Twitter.

I actually also had the idea to start a YouTube channel, but I know that's very time-consuming, so hell yeah, we'll see, it's not the prior 1. But let's spoil, it's a check, I've created a YouTube channel

Today, november 16th is the day I turn 41, exactly one year later. Here's the overview of what I accomplished 1 year after the initial idea.

My first package

I had the idea to build a package around the very popular permissions package of spatie. That permissions package makes it possible to define permissions for your application, but it has no gui, so the idea was to build a gui for it. The first name of my package was "permission-ui" but after some brainstorming I decided to call it "LLaodout inforce" as part of LLoadout. LLoadout, that stands for "Your loadout for Laravel, helping you kickstart your development process". The goal is to make it my learning-eco-system.

Other packages I've built

I've got a lot of ideas, big ones and small ones. Some ideas are implemented and packed into composer packages. At this moment I have created this packages that are available on packagist.

Laravel showsql : A Laravel package to output a specific sql to your favourite debugging tool. The supported log output is Laravel Telescope, Laravel Log, Ray, Clockwork, Laravel Debugbar and your browser.

Sajan : Sajan is a lightweight command line tool for webdevelopers

Smart : This packages enables the ability to serve file streams in a smart way

Database comparer : This tool compares two database structures and gives you the possiblity to generate a sql file or synchronize the database structure from source to target.

Laravel news dashboard tile : A Laravel News tile for the Spatie Dashboard

LLoadout inforce : With LLoadout Inforce you will kickstart your Laravel development when using Laravel Jetstream and Spatie permissions.

LLoadout components : With LLoadout Components you will pull in the best off class Laravel frontend components. We don't re-invent the wheel but we bring the best components together.

LLoadout refactor : Code base for refactorings I want to show you. Examples from real world projects.

My website

I had the idea to build a website. I love writing and are pleased to share some knowledge, tips and examples with other developers. With that goal in mind I've built dietse.dev. A full markdown driven website built with Laravel. I try to write some interesting stuff, sharing knowledge as main goal.

Another website I've built is refactor.lloadout.com , a website showing some refactorings from real life projects. It's open sourced and everyone can contribute refactorings.

Share information via Twitter

I follow a lot of content on Twitter and my goal is to share the good tips, using the most valuable content to write about in on my blog. I've done a lot of tweeting and sharing on Twitter for the last 12 months.

I've created a YouTube channel

Until now, I've created more than 10 videos, good for about 4000 views and 120 subscribers.
I love making video content, but it's very time-consuming.
My video on Laravel Livewire has been viewed more than 2000 times, and I'm very glad

about this achievement. If you're interested in the video content I've created it can be found here

Other contributions

I've contributed to some open source projects. Projects like the open source linux book from bobby liev, the tips lists from Laravel daily and the documentation of Blade uikit.

What the future will bring

I've done a lot, I think I don't want to create extra new channels but want to consolidate what I've started. Sharing new content, making new videos and hope to achieve my goal, giving something back to the community !

Sql knowledge will make you a better developer

Huh ? Why ? I have an ORM …

Yes indeed, you are using an ORM like Eloquent, and so do i. I use it as much as possible, I actually love it. But I do look at the queries the ORM generates and can interpret them because I know SQL. I've actually learned it at school 20 years ago and it is still very actual.

How to look into the SQL an ORM generates

I'll talk about Laravel, the framework I'm using every day. Laravel has some great third party tools that makes it very easy to look into that SQL. In fact, Laravel has its own tool to look into it , the tool called telescope.

I like telescope , but I have to admit that I don't use it so much. I'm a bigger fan of a debugbar like that of Barry van den heuvel. It has a tab to view the sql statements that were executed by your application.

And what is the advantage of that insight ?

It unlocks the possibility of speed monitoring on database level. Tools like telescope or barryvdh debugbar show you the amount of time a sql needs to be executed against your database. So if a query is slow you can copy it and run it against your database in a query tool like Querious. And than some SQL knowledge can be very useful.

Another pitfall can be detected very fast with those toolbars. The 1 + n problem. Assume you create a relation between a post and a user (author). Every post has an author. If you create an overview page showing you 100 posts with the corresponding user then you have to get the authors of that posts. While taking a look into the sql's ran it's possible that you detect a 1 + n problem. Meaning that for all the 100 posts an additional query is executed, resulting in 1 + 100 SQL statements in total. This can be solved by eager loading the author. Which will result in a total of 2 queries instead of 101.

The importance of indexes is underestimated.

And index is a mechanism that helps your database system to find data faster. Assume the relation between a post and a user. The id in the users table will be indexed automatically by your database, because it’s the primary key. But in the posts table the user_id (foreign key) will not be indexed automatically. Assume you have a big dataset it will be much harder for your database system to retrieve the related posts for a user is there is no index on the user_id in the posts table.

You have to take the same in account when searching data. It’s much harder for your database system to retrieve data from an unindexed field. I’ve seen situations where the addition of an index speeds up the query from 10 seconds to a couple of milliseconds.

So take care of your indexes !

That’s all ?

No it isn’t … What about scripts that need nitro speed … Scripts that store data into exports like csv files. Or maybe situations where you have to calculate sums on related tables. You can hand over the heavy lifting to your php script. Although, most of the time it’s a better idea to put some logic into your sql via subselects or logical operations in your sql. This makes it a lot easier for your php scripts and your server. Luckily we have the possibility to add raw statements to and eloquent builder, so you can combine the bost of both worlds. But, look out for sql injection while using raw statements if you rely on user input.

My opinion on sql knowledge nowadays

In my opinion there is a growing lack of sql knowledge. We’ve seen a huge shift to ORM’s the past 20 years , and i’m a big fan of it. But there is also a big shift in sql knowledge. There is a growing lack of underlying knowledge and this makes it a lot more difficult to keep focus on overal speed of an application. Especially in companies where there is no dedicated database engineer this can be a problem. Because in a development environment with little data everything is super fast. But what if there are a million records in that unindexed database.

The base concepts and pitfalls that come with joins, subselects, grouping aren’t learned thoroughly anymore at schools and that is something we really feel in the field.

The power of markdown

What is it

Markdown is, just like HTML a **markup language**, it's the language you might know from the README.md in **almost every open source Github repository**.

The overriding design goal for Markdown’s formatting syntax is to make it as **readable as possible**. The idea is that a Markdown-formatted document should be **publishable as-is**, as plain text, without looking like it’s been marked up with tags or formatting instructions.

Why should you use it

Markdown is extremely **easy to write**, it has no tags and with a very **little of markup knowledge** you can write all your texts. It's also extremely simple to **transform markdown to html**.
So markdown is the perfect language for **writing prose**. It's the perfect tool for writing blogs like this or books.

Markdown use cases

Markdown is mainly used to **create documentation**. It's a simple text format with **simple markup**. That format can easily be transformed into **html and styled with css**. So it's often used for **website** creation or **e-book** creation, all because of its simplicity and ease of use.

If you're reading this blogpost, well, then **you are actually reading markdown**. This site is almost completely markdown driven. The markdown files are a simple format to write in, and it has te possibility to serve information in a multi-purpose way.

What is used on this site to parse the markdown to html

On this site I'm using **spatie/laravel-markdown** to convert the markdown files to html. The meta information added to the markdown files is parsed with **spatie/yaml-front-matter**. Mohammed Said, a core member of the Laravel team has written a library called **Ibis**, this library makes it possible to convert your markdown to a pdf book.

The markup

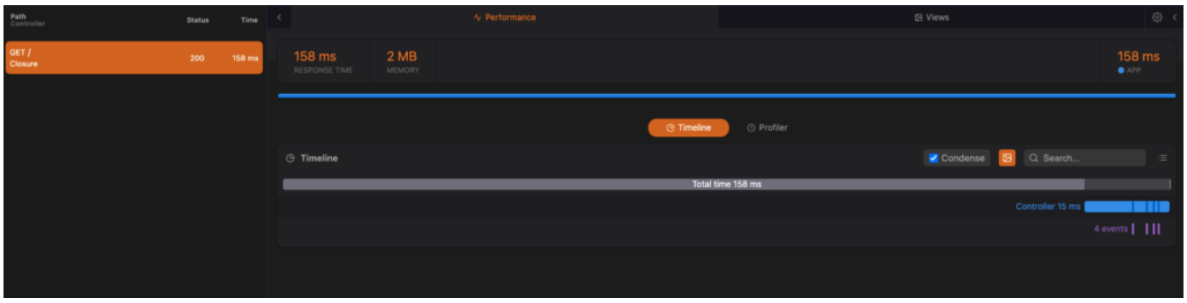
Markdown has some simple tags like # , ### for headings , ** for bold text * for italic text. A complete overview of these tags can be found on this site!

This is what you can do with clockwork

What is clockwork ?

As stated on The clockwork github repo , Clockwork is a development tool for PHP available right in your browser. Clockwork gives you an insight into your application runtime - including request data, performance metrics, log entries, database queries, cache queries, redis commands, dispatched events, queued jobs, rendered views and more - for HTTP requests, commands, queue jobs and tests.

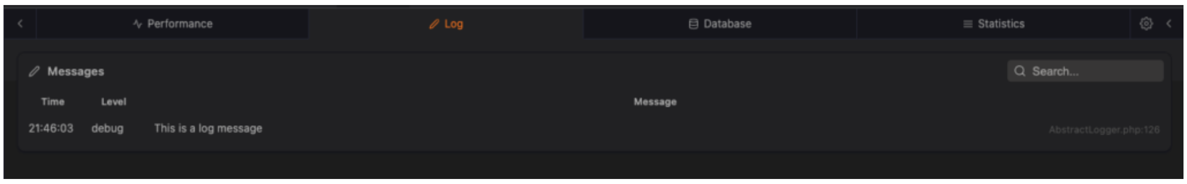
It's as simple as it gets , you install it via composer , install the browser extension, and you are set , for free.



Log to a dedicated window

With clockwork it is possible and super easy to log messages to a dedicated tab in your developer tools

```
clock('This is a log message');
```



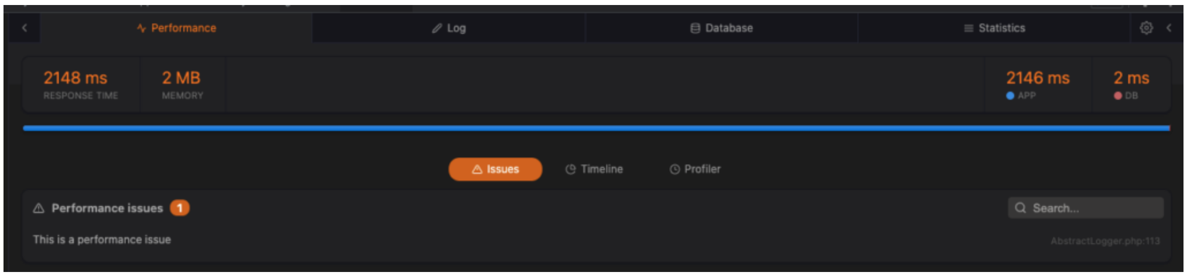
```
Time      Level     Message
21:46:03  debug     This is a log message          AbstractLogger.php:126
```

the output of the clock command in the log tab of clockwork

Performance issues

it is possible to log to the performance issues tab

```
clock()->info("This is a performance issue", ['performance' => _true_]);
```



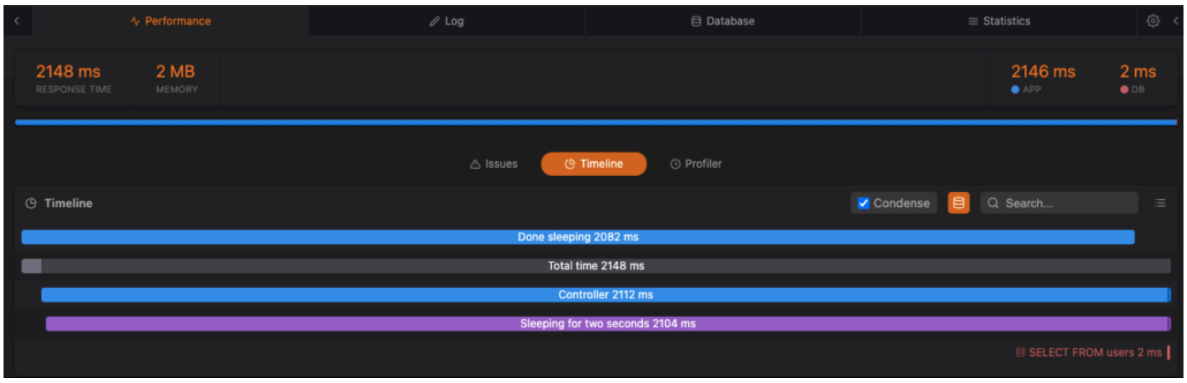
```
This is a performance issue                                AbstractLogger.php:113
```

This log is visible in the issues section of the performance tab

Measure performance

With clockwork it is very easy to measure the time it takes to execute some code

```
clock()->event('Sleeping for two seconds')->color('purple')->begin();
sleep(2);
clock()->event('Done sleeping')->end();
```



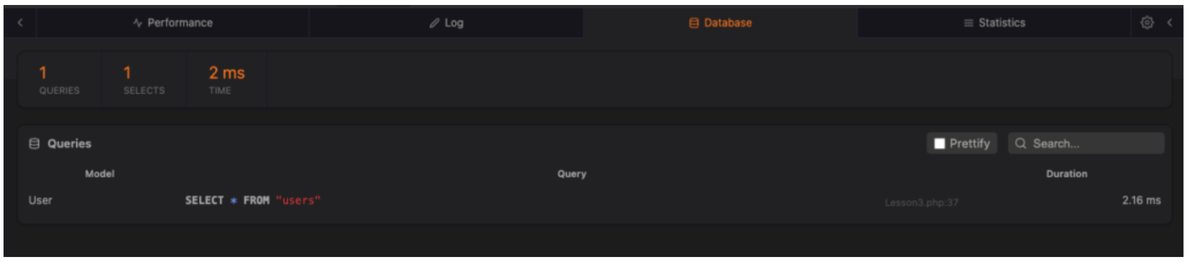
```
Done sleeping 2082 ms
Total time 2148 ms
Controller 2112 ms
Sleeping for two seconds 2104 ms
SELECT FROM users 2 ms
```

the measurement is shown in the timeline section of the performance tab

Show your database queries

With clockwork it is very easy to visualize your database queries and the time it takes to execute your queries against your database.

Every query will be logged to the database tab. It shows your total query time and the time per individual query.

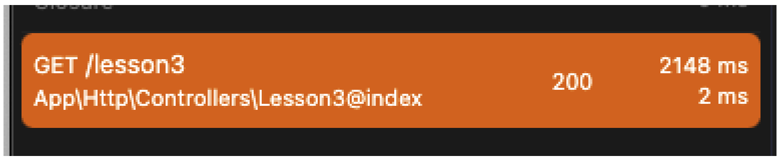


1 QUERIES | 1 SELECTS | 2 ms TIME

Model	Query		Duration
User	`SELECT * FROM "users"`	Lesson3.php:37	2.16 ms

Clockwork points to your controller

It's often not so clear which controller method a route is pointing to , with clockwork this is clear visualised on the left hand side of the toolset.

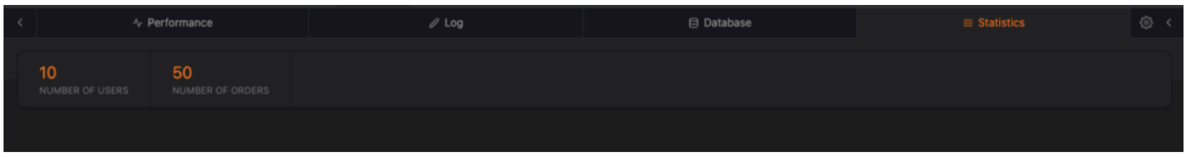


GET /lesson3 200 2148 ms
App\Http\Controllers\Lesson3@index 2 ms

Create custom tabs

It's possible to create custom tabs and put the info in it you want, creating your own datasets in clockwork.

```
$cart = clock()->userData('Statistics')->title('Statistics');
$cart->counters([
    'Number of users' => 10,
    'Number of orders' => 50,
]);
```



10 NUMBER OF USERS | 50 NUMBER OF ORDERS

What about collections

What are collections ?

Collections are a Laravel **wrapper** for **arrays** to work with data in a **fluent and expressive way**. The collection methods are **chainable** and give you a very **powerful toolset** for working with data.

Methods

Collections provide their power via **self explanatory methods.** The full list of available method can be found in the documentation.

An example of a method is the "sum" , making it extreme simple to take the sum of all elements in an array

```
collect([1, 2, 3, 4, 5])->sum();
```

Chaining

An extra power of collections is that they can be chained , which means that you can execute a chain of actions on a collection retreiving the results in one code flow. Like in the example below with groupby , map and sum.

Why I think you should use collections

Every method on a collection uses underlaying array logic. So you could achieve the same result when using arrays. But collections provide you with a **much more discriptive way of programming**. The code is mostly **self explanatory** and much easier to understand. Not only for you as a developer, but also for team members reviewing your code or adapting your code. I'll give you an example. Assume you want to get the sum of the price of the lines in an offer grouped by the group the lines are in.

You could achieve it like this with an array

```php
$totalPerGroup = [];
if (count($offer->lines) > 0) {
    foreach ($offer->lines as $line) {
        if (isset($totalPerGroup[$line->group])){
            $totalPerGroup[$line->group] += $line->price;
        }else{
            $totalPerGroup[$line->group] = $line->price;
        }
    }
}
```

This is a perfect working example , but it can be done much more readable and descriptive if you use collections

```php
$totalPerGroup = collect($offer->lines)->groupBy('group')->map(
    fn($group) => $group->sum('price')
);
```

By readling this code it explains what it does , giving you the extra advantage that you have to write less documentation in your code.

Easily expandable

Collections are "macroable", which allows you to add additional methods to the Collection class at run time. With this power you can write your own collection methods making your custom logic even more readable and reusable.

My top used collection methods

If I think about which methods I use the most I see myself using these : filter, where, transform, sum, each, keyBy, groupBy, sortBy, toArray, values . Of course I use more than only these , but these are my top 10.

Database speed killers I often see

Searching for performance enhancements

I’m building web applications for somewhat 20 years now , and i’ve been dealing with many speed issues through time. Issues that made me a better developer, but issues I often see coming back in code examples and pull requests from other developers. Here are some tips how to avoid this speed issues.

Retrieving related data without eager loading

Most of the time we have to deal with related data. I’ll give an example … A user has posts and post have comments. These relations will be defined in your eloquent models with hasMany relationships. Ok so far so good.

Now when selecting this data, I often see the following construction , what’s looks good at first sight.

```
$users = Users::all();
foreach($users as $user){
    foreach($user->posts as $post){
        echo $post->content."<br />";
        foreach($post->comments as $comment){
            echo $comment->content."<br />";
        }
    }
}
```

Ok well done , we are showing all the posts and comments for all users. Job done … No not really. Here we deal with the N+1 problem. For every user (1 sql) , all posts (a sql per user) , and comments (a sql per post) will be fetched from te database. Assume we have 10 users with 10 posts and 10 comments we will have 1 + (10 * 10) sql statements. So we execute 101 queries against our database. If we have 100 posts with 10 comments … 1001 statements ?

This can be done better with eager loading, by only changing one line of code.

```
$users = Users::with('posts.comments')->get();
```

This will result in the same datasets, but it will be done in 3 queries … 1 per model.

More info can be found in the Laravel docs here

We like stars (*) , don’t we

Assume you have a huge database table , with a lot of columns and you perform a select statement on it retreiving models. You will typically perform some search logic and perform a ->get() on to retreive your models. This will return all columns from your database loading a massive dataset in memory. It’s much better to specify the columns you want by add an array of fields you want to return. For example ->get(‘title’,’comment’) , this will only return the data you want, using less memory.

A database index ? Que !?

unfortunately , many developers don’t think about indexes. Indexes make your database much more performant , it tells your database which data it should keep top of mind so it’s faster to receive. This is in my opinion the most overseen problem when dealing with big datasets. And why it’s easily overseen is easy to explain. We mostly develop new features with smaller datasets, with not that than much related data. When going in production and when datasets grow this is the number 1 speed killer. Slow search queries , bad indexed relationships are big speed killers , try to avoid them.

Eloquent for president !

Eloquent is great , I love eloquent , I adore eloquent ? . But not always. If I need raw nitro speed I mostly use raw query statements on the query builder. This will execute my queries directly against the PDO layer. Data will not be loaded into model per definition. The most common use case for this approach is reporting. Here we combine data from several tables and collect that into arrays to pass to an Excel builder. This can perfectly be done via raw statements with joins return arrays of data. You really have to try it if you need raw speed.

More info on raw query statements can be found here

Gimme all !

Off course you want all your data , but sometimes it’s a better idea to only query things in parts , in chunks. This because of memory optimisations. If you want to perform actions on , lets take , 1 000 000 records , and you fetch these models in 1 call you might hammer you servers memory. Use the chunk method for these use cases and you will keep your memory usage under control.

My opinion on documenting your code

Do not overdocument your code

In my opinion it is not very useful to extremely document your code. Your code is computer code, indeed, but write it for humans. If you write code that reads as a book then your code will be self-explanatory. A good starting principle that will save you a mass of time.

Naming stuff

Use good class names, method names and variable names. They come at no cost and are the base of a good codebase that needs less comments.

I don’t think there is much added value in writing extra documentation for the code block shown here. The method name is telling you what it does and the variables are also loud and clear. These are very small code blocks put the principle can be applied to every code part of your software.

Good :

```
public function calculateTotalOrderValue(Order $order){
    $totalValue = $order->lines->sum('price');
    return $totalValue;
}

public function calculateSurface($width,$height){
    return $width * $height;
}
```

Bad:

This example is an example of bad code … It does the same as the code mentioned higher, but it is totally unclear what it does without interpreting the code itself.

```
public function calculate($firstparam){
    return $firstparam->lines->sum('price');
}
```

Don’t misunderstand me ?

You have to write documentation , it’s a must … But do it the good way and don’t repeat yourself. Write added value … Don’t write what your code tells you. I prefer better code above more documentation.

Write documentation for the bigger picture

Instead of documenting all methods you better learn to write self-explanatory methods. That will save you a lot of time.

It still is a good idea to document what you’re doing with a higher overview. You can explain what the total flow of the code does, so you can get the bigger picture by reading these docs.

Format your code

Code formatting makes your code more readable. Tools like PhpStorm have built-in code-formatting tools. Things like aligning key value assignments , formatting with equal spaces … makes your code more readable and comprehensible.

Profiling and debugging tools

First, the hassle

Everyone knows the problem of debugging an application. The most popular way to do it in php is using the dump and die method, better known as dd in Laravel. We output something to the screen and exit our application. So you can see the output of a method and stop the execution of your program.

That's not the best thing to do though, your program stops running and can only dump in one place.

There are some handy tools

This can be done better by using some handy tools. Tools that don't necessarily stop the execution of your program, so you can output content on different places in your application. There are some nicely crafted community driven profiling tools to help you as a developer.

Measure

How long does a query take ? How will the mails you sent look like ? how can I debug easier than the var_dump method. How many memory is used.

I'll show you 4 tools in a video

In this video I give you a very brief introduction of 4 tools. Laravel Telescope , Barryvdh Debugbar , Laravel Ray and Clockwork. I’ll show you the setup of it. I will also show you where you can see the queries executed against your database. We check some data that is sent to our views and test a simple mailable.

Take a closer look to a single sql statement in Laravel

Showing the sql statements processed by your application ?

Most of our applications are database driven. When using Laravel you are using Eloquent and the Query Builder all the time.
These very handy built-in tools are capable of transforming your code to database queries that are executed against the database of your choice. In my case this is mostly MySql.

We often come in the situation where we have to enhance performance or troubleshoot a query. In these situations you need to take a look into the queries that are executed. Telescope is an example of a tool created by the Laravel core team that is capable of doing that. The tools of my choice are itsgoind/clockwork or barryvdh/debugbar. These tools give you a dedicated tool into your application or your developers tools that can be used without leaving your application.

All of these tools show you all the statements that are processed by your application, so if you perform 20 query in a request, all these queries will be shown. This is not always the best solution, if you are debugging something then you want to draw attention to one single part of the code, and this might be a single query.

To solve this issue I've created a composer package, I'll give you a short tour in the next chapter.

Laravel showsql

Laravel showsql is a Laravel package to output a specific sql to your favourite debugging tool, your browser or your log file.

Use case

You often want to draw the attention and look into one single sql while you are developing. You can look up your sql in your favourite debugging tool in the sql tab , but most of the time your sql is not the only sql executed ... So the searching begins. With this package you can add `showSql()` to your QueryBuilder and the single sql will be outputted to the logging of your debug tool.

The supported log output is Laravel Telescope, Laravel Log, Ray, Clockwork, Laravel

Debugbar and your browser. By default, showSql will try to log to Ray, Clockwork or the Laravel Debugbar if one of them is installed. If all installed it will be output to all. If you want your own log implementation you can pass a callback to showSql.

If you want to change this behaviour you can publish the config file and change it.

Installation

```
composer require dietercoopman/laravel-showsql --dev
```

Examples

```
# With the Eloquent Builder

Menu::showSql()->get();

Menu::whereId(1)->showSql()->get();

Menu::whereHas('status')->showSql()->get();

# With the Query Builder

DB::table('menus')->where('id', '=', 10)->showSql()->get();

DB::table('menus')->join('statuses', 'statuses.id', '=',
'menus.status_id')
                 ->showSql()
                 ->get();

# With a callback

$callback = function(string $sql){
  Log::info($sql);
};

DB::table('products')->where('id', '=', 1)->showSql($callback)->get();
```

Writing command line tools with php

Why would you build your own command

We use the commandline a lot, and maybe you have a great idea for building your own command line program. It can be a tool like composer or php artisan , things that make the life of developers easier and are real time savers. So let's go and build that tool, I'll explain you some possibilities.

How to build ?

The most well know way of building command line tools is building the tool with bash scripting, the language of the commandline. That's a possibility if you like it , but I don't prefer it let's figure out why not ...

Why not in bash scripting

For me, it feels much more comfortable to program in PHP, maybe it's because of the lack of knowledge of the language but bash scripting is much more cumbersome to write in than PHP. Php projects can be heavily structured and yeah, it's the language I'm familiar to work in, so it works better for me. I've created Sajan version 1 in bash scripting and to be honest if I open that codebase now I always have to figure out what is does. It's not possible to structure all the logic in classes and services (as far as I know), so the bloat grows fast.

The technology to use

You can write your PHP command's in vanilla PHP, but that's not my choice. By using a tool called Ibis, created by Mohammed Said I discovered the symfony console. That's a symfony component built to help you create command line tools. It has a nice Api and provides you with the possibility to use colors, questions, choices , all out of the box.

An example

This is a command created for my open source tool Sajan, it's a command in Sajan to show the ip address of your computer.

I show you an extract of the code but please feel free to take a look in the full source code on GitHub, that will learn you a lot.

In the first section, the configure section you can specify the name of your command, I've set a description and an alias. So this command can be fired by calling `sajan ip:lan` or `sajan il`. The command will output something like `Your lan ip address is : 192.168.3.101`

You can see this is a fluent way of bash programming using the symfony console component.

```php
<?php

namespace Dietercoopman\SajanPhp;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Question\Question;
use Symfony\Component\Process\Process;

class IpLanCommand extends BaseCommand
{
    /**
     * Configure the command.
     *
     * @return void
     */
    protected function configure()
    {
        $this
            ->setName('ip:lan')
            ->setDescription('Get your lan ip address')
            ->setAliases(['il']);
    }

    /**
     * Execute the command.
     *
     * @param InputInterface $input
     * @param OutputInterface $output
     * @return int
     */
    public function execute(InputInterface $input, OutputInterface
$output): int
```

```
    {
        $ip = Process::fromShellCommandline("ifconfig | grep \"inet \" |
grep -Fv 127.0.0.1 | awk '{print $2}'")->mustRun()->getOutput();
        $output->writeln('<fg=yellow>Your lan ip address is :
</><bg=red> '.trim($ip).' </>');

        return 0;
    }
}
```

Image handling with intervention/image

Very nice and simple api

Image intervention Image intervention provides us with a fluent and simple api. It becomes very easy to transform images in several ways. The most popular use case for images manipulation is resizing. I'll give you a few examples on how to handle it.

The source code of this tip can be tested via the Lloadout tutorials GitHub repo. If you better like watching video you can watch my YouTube video on image intervention.

Loading an image

The make command provides you with a simple function to transform images to image objects. You can load urls, image streams, paths.

This example loads an image from an external resource and returns it to the browser as a jpg.

```
$img =
Image::make('https://raw.githubusercontent.com/LLoadout/assets/master/LL
oadout_tutorials.png');
return $img->response('jpg');
```

If you want to rotate your image you can call rotate on the image with the number of degrees you want to rotate the image on.

```
$img->rotate(90)->response('jpg');
```

Resizing an image

If you want to resize an image then you call the resize method with the width and height arguments, this will force the image to be returned in the provide dimensions (not keeping the aspect ratio)

```
return $img->resize(300,400)->response('jpg');
```

resizing keeping aspect ratio

Giving your image an absolute with and height is often not a very good idea, your aspect ratio will get lost and images will look very weird. So its possible to keep the aspectratio of the image.

```
return $img->resize(200, null, function ($constraint) {
    $constraint->aspectRatio();
})->response('jpg');
```

Prevent possible upsizing

Another side effect of image resizing is possible upsizing, images get blurry by upsizing so it's better to prevent it. This can easily be done.

```
return $img->resize(null, 1000, function ($constraint) {
    $constraint->aspectRatio();
    $constraint->upsize();
})->response('jpg');
```

Saving a processed image

If you want to handle a file and save it somewhere you can store it on your file system , eventually using Laravel disks if you want to.

```
return $img->save(storage_path()."/lesson1/lesson1.jpg",80,'jpg');
```

Colorizing images

Sites using a lot of images in different colors can look overwhelming , that's why it might be a good idea to transform the images to be grayscale.

```
return $img->greyscale()->response('jpg');
```

Maybe you want to give all your images a specific color

```
return $img->colorize(-100, 0, 100)->response('jpg');
```

Writing text on an image

In some cases you might want to add text to your images. A common use case for text on an image is a watermark stamp.

```
return $img->text($_GET['text'] ?? 'LESSON 1', 20, 40, function($font) {
    $font->file(storage_path().'/lesson1/Raleway-Medium.ttf');
    $font->size(30);
})->response('jpg');
```

Open source

One of my goals is to create some awesome tools and packages for other developers , this is an extract of tools I'm building

Laravel showsql

A Laravel package to output a specific sql to your favourite debugging tool, your browser or your log file.

Use case

You often want to draw the attention and look into one single sql while you are developing. You can look up your sql in your favourite debugging tool in the sql tab , but most of the time your sql is not the only sql executed ... So the searching begins. With this package you can add showSql() to your QueryBuilder and the single sql will be outputted to the logging of your debug tool.

Installation

You can install Laravel showsql via composer, more info on the Laravel showsql github repo

Lloadout

LLoadout aims to be your loadout for Laravel, helping you kickstart your development process LLoadout is made up of four pillars: learning , building , automating and optimizing. Everything about LLoadout can be found on the lloadout website

Tutorials

With the video tutorials we try to optimize the learning process with short valuable content of approximate 5 minutes.

Components

With LLoadout components you will pull in the best of class Laravel frontend components. This can drastically speed up your frontend development.

Inforce

LLoadout Inforce is a toolset to kickstart your application on top of Laravel Livewire, Laravel Jetstream and Spatie Permissions. LLoadout is created using the TALL stack.

Refactor

LLoadout refactor is a site thats show you some nice refactors, giving you valuable code examples.

Sajan

What is it ?

Sajan is a lightweight command line tool for webdevelopers, it is written in php .

The tool provides you with shortcuts for multiline commands or opens a world of hidden oneliners.

Requirements

Sajan is a shell tool tested on MacOs desktop and Linux servers

Sajan uses tools like brew, git, npm, node and composer.

For us brew is the best tool to install these programs, so we depend on it. If you don't have it, you can install it easily via sajan.

Installation

You can install sajan via composer, more info on the sajan github repo

Smart

What is it

Smart is a blade component for easy image manipulation. Want to serve private hosted images without the need to code your own logic ? Want to resize your images before sending them to the browser, without extra coding ? Then smart might be your buddy !

This package makes it possible to

serve images that are not public accessible without coding resize images without coding resizing public hosted images without coding automatically cache your images

You can install smart via composer , more info on the smart Github repo

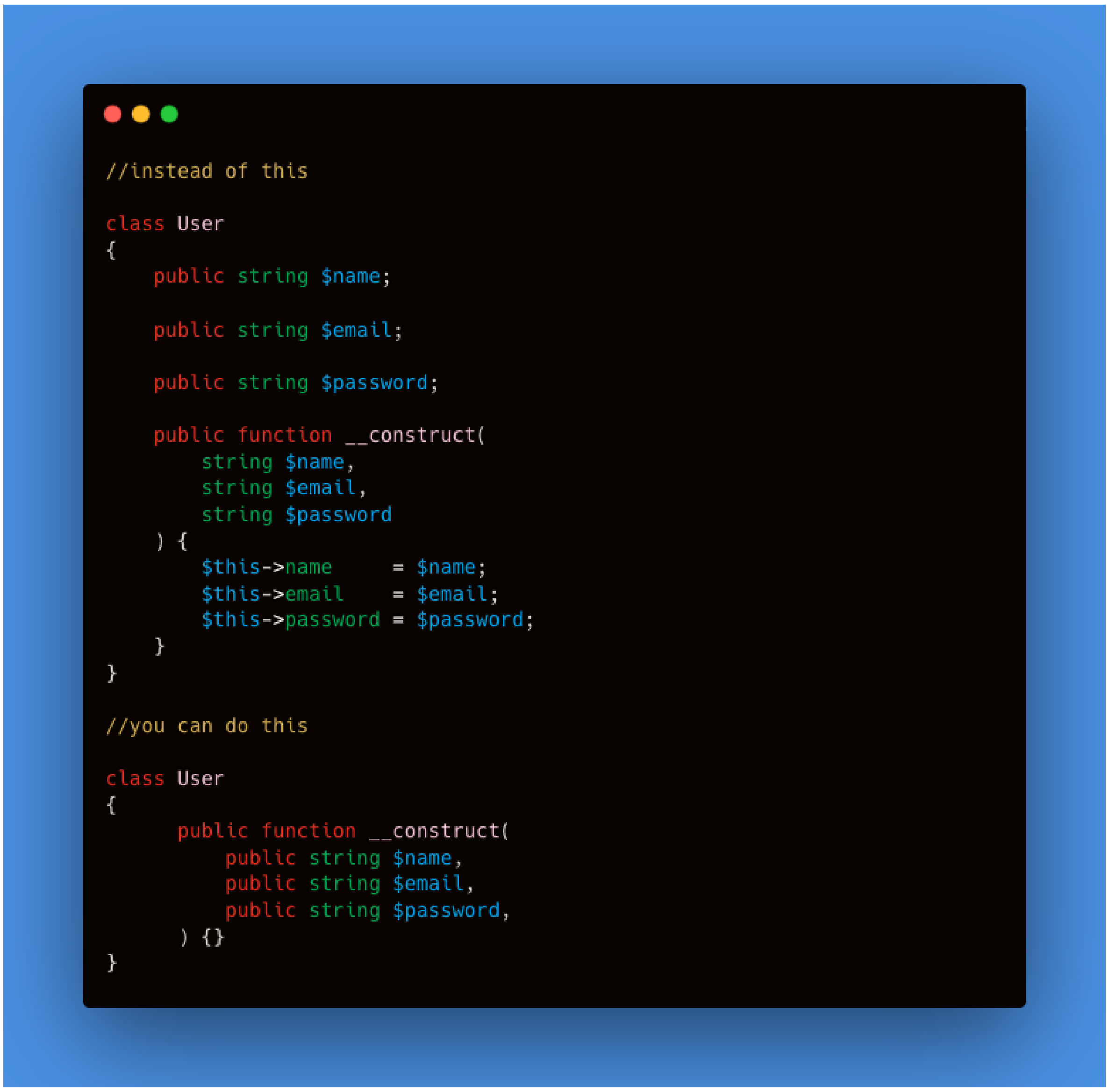
Other projects

I have some other little projects that can be found on my Github page

Quicktips

The power of constructor property promotion in php8

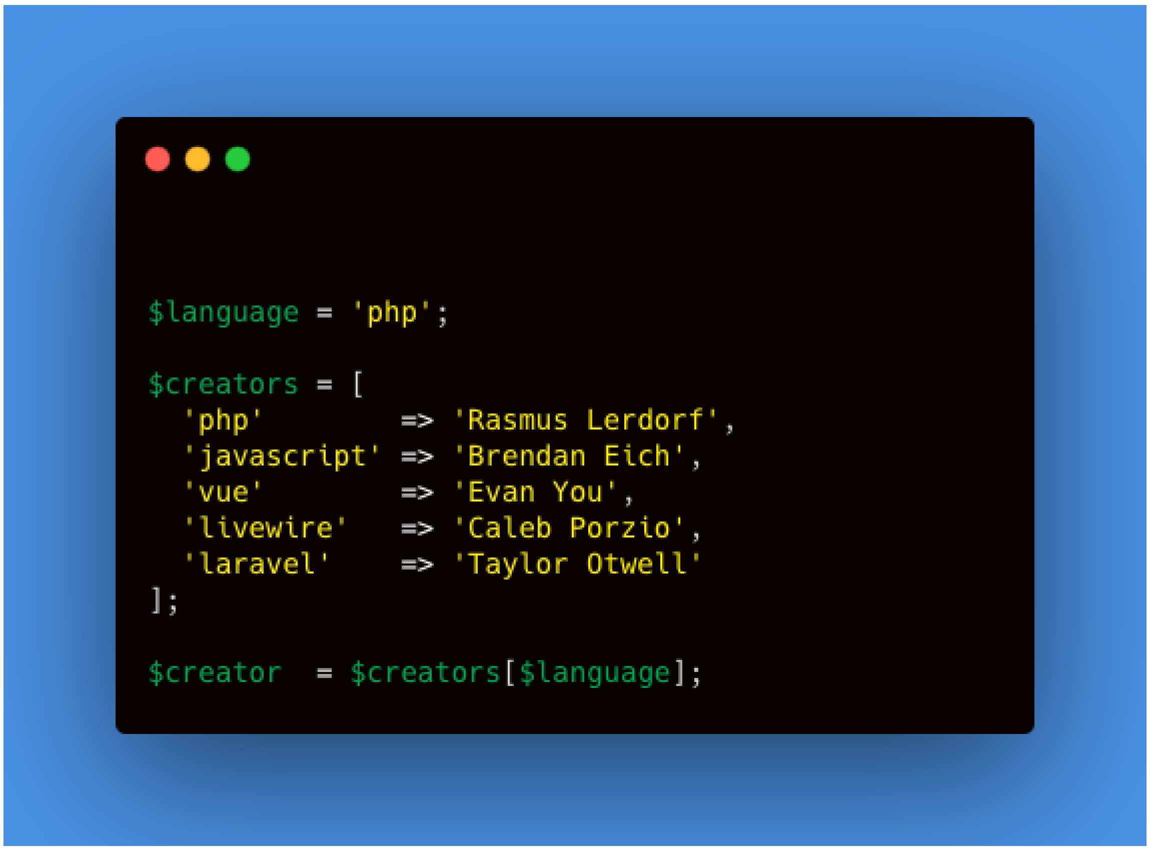
We no longer need to assign our properties



```php
//instead of this

class User
{
    public string $name;

    public string $email;

    public string $password;

    public function __construct(
        string $name,
        string $email,
        string $password
    ) {
        $this->name     = $name;
        $this->email    = $email;
        $this->password = $password;
    }
}

//you can do this

class User
{
      public function __construct(
          public string $name,
          public string $email,
          public string $password,
      ) {}
}
```

The power of arrays as decision makers

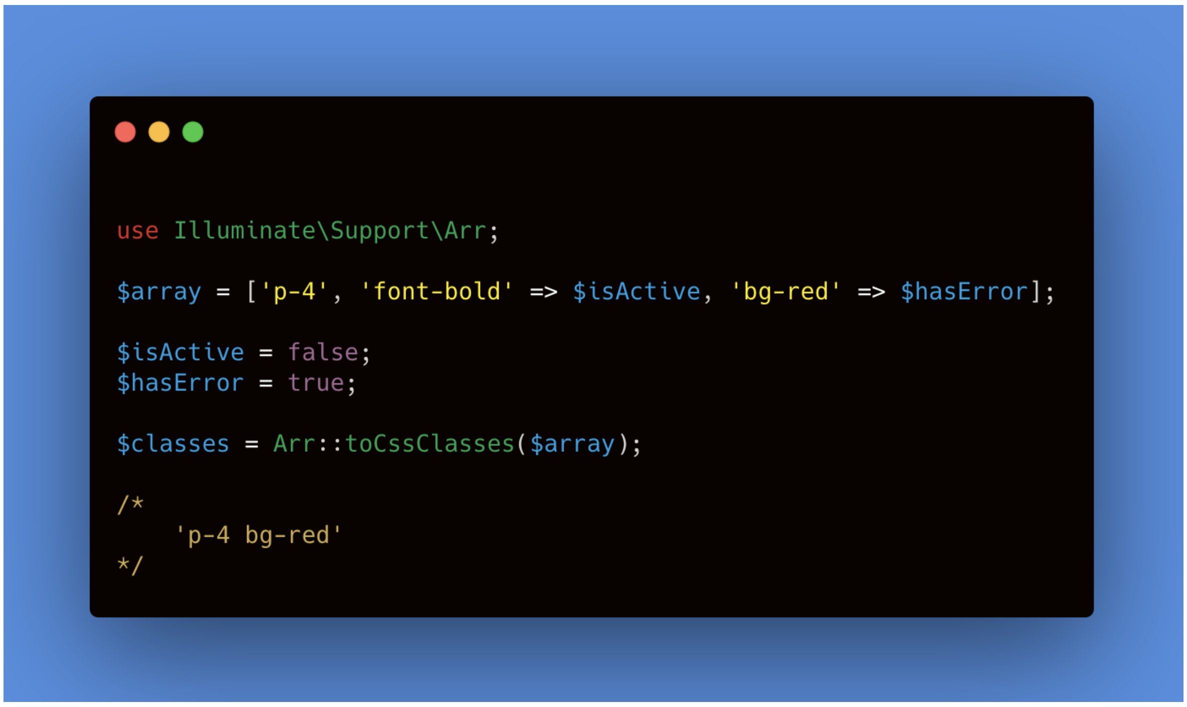
No need for an if , switch or match syntax ... only simple arrays ... super readable



```php
$language = 'php';

$creators = [
  'php'        => 'Rasmus Lerdorf',
  'javascript' => 'Brendan Eich',
  'vue'        => 'Evan You',
  'livewire'   => 'Caleb Porzio',
  'laravel'    => 'Taylor Otwell'
];

$creator  = $creators[$language];
```

Transforming an array to css classes

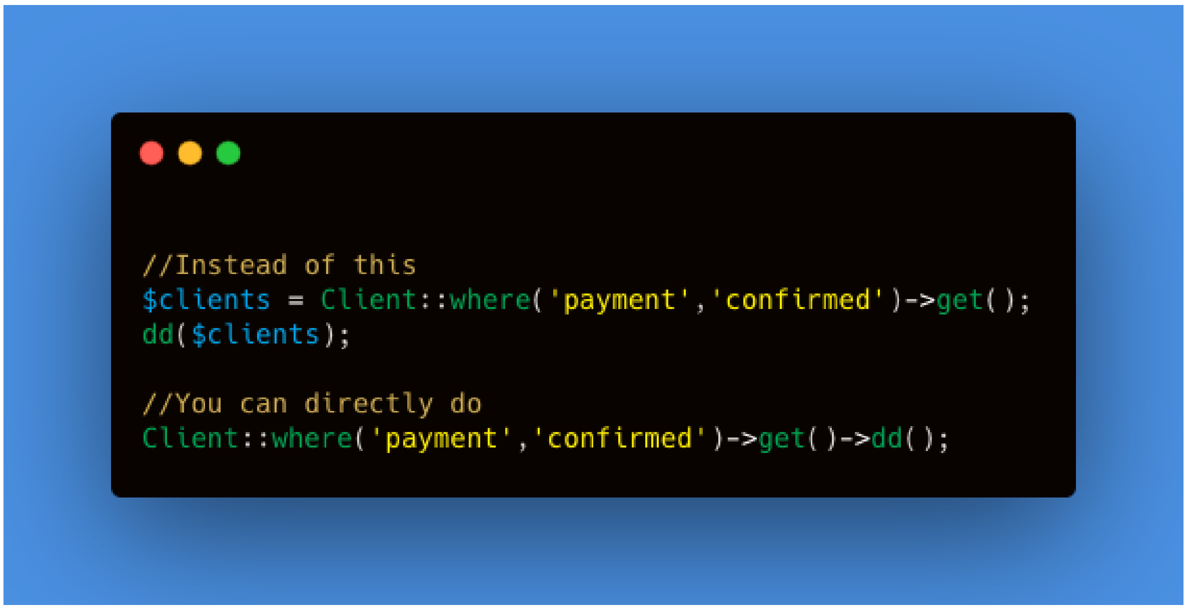
tip by Laravel documentation



```php
use Illuminate\Support\Arr;

$array = ['p-4', 'font-bold' => $isActive, 'bg-red' => $hasError];

$isActive = false;
$hasError = true;

$classes = Arr::toCssClasses($array);

/*
    'p-4 bg-red'
*/
```

Inline dd

tip by Laraibi Mehdi



```php
//Instead of this
$clients = Client::where('payment','confirmed')->get();
dd($clients);

//You can directly do
Client::where('payment','confirmed')->get()->dd();
```