CHAPTER 1

Letter From The Editor

Thank you for your support of Laravel News and welcome to the first issue of the Laravel Journal. This is a new medium for me and I wanted to create something unique for you and bring it in the best reading experience possible. Based on a quick Twitter survey most people liked epub because it's the most flexible and fonts and colors can be changed to suit your needs.

In this issue I have the announcement of Laravel Spark, an interview with Jesse Schutt, a tutorial on how to write a readme file, and a community corner to highlight some of the cool resources that have been released this past month.

I hope you enjoy it and I'd love your feedback. You can always reach me at eric@ericlbarnes.com

CHAPTER 2

Laravel Spark

Laravel Spark, the Laravel package that provides scaffolding for subscription billing, is now officially released and available for everyone.

August of 2015 at the Kentucky Center hundreds of developers sat waiting as Taylor Otwell took the stage for the Laracon Keynote talk. He kicked if off talking about finding inspiration from the movie *Jiro Dreams of Sushi*.

"Jiro would dream of sushi, wanting to elevate his craft. One could assume this gives him a deep and satisfying meaning to life", Taylor recalled, "we share a lot of the same characteristics, and one big item is flow."

Flow, also known as the zone, is the mental state of operation in which a person

performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment in the process of the activity.

Taylor continued by talking about how Laravel aims to make the pathway to flow as easy as possible by allowing you to focus on the app you are building, not the mundane boilerplate.

Some of the examples include Homestead as a development environment, Forge and Envoyer for production deployments, and all the other tools Laravel provides like Blade, Eloquent, and Elixir.

This lead to his newest creation Laravel Spark, which aims to make creating a working SaaS (software as a service) app painless.

The keynote continued going through a demo and announced the alpha would be available the following month. Then all went silent.

Having just launched Forge and Envoyer, Taylor knew how much of a burden building out subscription billing is. It needs a payment gateway, full member management, subscribing to plans, subscription swapping, subscription canceling, pro-rating, coupons, discounts, teams, invites, and that is just the bare minimum feature set. You also have to add routes, views, JavaScript, and more to get this all working.

What Spark does is combine all this into a single package that you can add to a Laravel app. Define your subscription plans, set your company address, add Stripe data, and deploy.

Development of Spark continued behind the scenes, and little teasers of the official version started being leaked to social media at the end of February. Then on March 3rd, Taylor announced the official release will not be free.

During the keynote, Spark was released as a free open source product, and this pivot caused a mix of both positive and negative reactions. Most were understanding of the change in direction.

The #sparkwatch teasers didn't stop, and then a super specific launch date came out. April 19th at 10 AM CET. As that date started approaching Spark became feature complete and a beta launched Friday, April 15th.

The pricing is $99 for a single license and $299 for an unlimited. With one license you can develop as many sites as you want locally and it only goes into effect once you deploy to a server.

CHAPTER 3

Unplugging With Jesse Schutt

Jesse Schutt is a developer for Zaengle Corp where he works on a mixture of projects between Laravel, Craft CMS, and Statamic. Before making the jump into development Jesse was a media director for ten years at a year-round Bible camp in northern Wisconsin. He's a husband, father of six kids, and someone full of inspiration and quality advice.

Hi Jesse, Can you tell us about your typical workday?

Whenever possible I like to begin before the kids wake up. My typical routine consists of starting the coffee pot and getting an early jump on the day. Since I work from home and we have 6 children, I need to find stretches of quiet time to focus on development and the early mornings offer it.

Most of my development time is spent on Laravel work with some Craft CMS customizations

tossed in here and there. Before I became a full-time developer I did both backend and frontend beginning in the mid 2000s, but now I've specialized into PHP, primarily with custom Laravel apps.

Do you have any hobbies?

Ever since I was a young child I've lived in rural areas; some might call them "the middle of nowhere"! The lack of commercial entertainment caused me to find my own ways of entertaining myself. Most of them included being outside, whether camping, hiking, fishing, or hunting.

During the winter months I spent a fair bit of time woodcarving. My parents had signed me up for a carving class and I enjoyed making figurines or small fish and animals.

As I got older and was able to start using my dad's tools I became interested in building projects that were functional as well as entertaining to make.

What is the difference in woodcarving and woodworking?

I suppose you could call woodcarving a subset of the broader topic of woodworking. It typically doesn't require much for tools, aside from a few carving knives. And it doesn't require much space either!

Woodcarving projects can be anything from a small Santa figurine, to a duck decoy, or a relief carving that will end up built into a piece of furniture.

Spoon carving has become quite popular recently. All it takes is a piece of wood, typically birch, or some kind of fruitwood, and a bit of time to whittle it into shape. My wife has a handful of spoons I've carved and uses them regularly in the kitchen.

How did you originally get into woodworking?

My dad is handy, always tinkering on a project around the house. When I was in high school he worked on small building projects for the local lumber yard and I would help out in the afternoons. We joked that our construction name would be "Stupid and Clumsy Builders"... I'll let you guess which one I was ;P

Although I had access to tools whenever I wanted, and I learned the basics from my dad, I really didn't engage in fine woodworking (cabinetry, box-making, detail-work) until a number of years later. My wife and I had bought our first house in 2009 and I realized that I finally had some space to keep my own tools. A few months later and a sweet Craigslist deal found me with a pretty well-equipped shop.

As someone who has never built anything, what is the biggest difference between tech and woodworking?

Yeah, woodworking is an ancient practice. The Bible says that Jesus was a carpenter! (That must make it the most holy of hobbies, right??)

Joking aside, there is something special about looking at a piece of wood, and seeing potential beneath the surface. As a developer I enjoy bringing order and structure to bits of code. The same goes for woodworking. I find great satisfaction in taking raw material, shaping it, finishing it, and turning it into a completed project.

That satisfaction is very similar to the feeling I have when implementing a clean, well-tested solution to a code scenario. In code, I'm organizing the virtual, while in woodworking I'm organizing the physical. I believe we were made to be creators, whether with our hands or brains.

There has been a resurgence of woodworking in recent years. I'm guessing this has to do with how much we've moved our lives into the digital realm, which can be draining if you live there too long. Woodworking is a way to use modern technology, such as high-quality tools, with an intimate connection to the physical.

The skills required to be a good woodworker are very similar to being a good developer. We need precision, forethought, the ability to picture a project in your head, and solid problem-solving skills. Quality woodwork is clean, even in the joints that aren't visible. Quality code considers all the scenarios, not just the most obvious ones.

Do you have to keep learning and studying the craft or can you learn everything and then use your imagination to build anything you want?

Like any venture in life, you need a certain amount of experience combined with just the right amount of inspiration. As I look over my growth as a woodworker, I think the best teacher has been failure.

It can be extremely challenging to work with wood, as it can crack, warp, or bend. Humidity and temperature all cause movement in boards. Every time a joint doesn't come together cleanly, or a piece doesn't fit as it should, I try to make a note in my mind for the next time I'm presented with a similar situation.

As any good hobby should, I enjoy spending my spare time reading on new techniques. So learning is a significant part of developing as a woodworker.

However, experience is a wonderful teacher as well. Don't be afraid to try something new!

Most developers have hobbies that are computer related. Do you find it helpful to have a hobby that allows you to unplug?

One of the things that I love about woodworking is that it allows me to do something with my hands! Since the majority of my day is consumed staring at a screen with hands on the keyboard, I find that breaking out of code-brain can be challenging.

For example, it takes me a bit of time to get into the mental space necessary to effectively

program. Woodworking is a happy medium, where I exercise a blend of creativity and logical, systematic thinking, while engaging with the physical realm. Therefore, there are many days where I will program for the majority of the day, but move into the woodshop where I will prepare to reconnect with my wife and kids. I've found it good for our relationships if I can gently make the transition out of "code-brain". Woodworking has offered me that avenue.

Plus, at the end of a project I have something that can be given away as a gift, or proudly displayed in my home!

What advice would you give someone interested in getting into woodworking?

Everyone learns differently, but I've found a few things that may be helpful if a person wants to dig into woodworking:

Search the #woodworking hashtag on Instagram.
Woodworking is extremely popular on Instagram, with some of the leading content producers having more than 50k followers. The community is also very friendly and will readily engage with you.

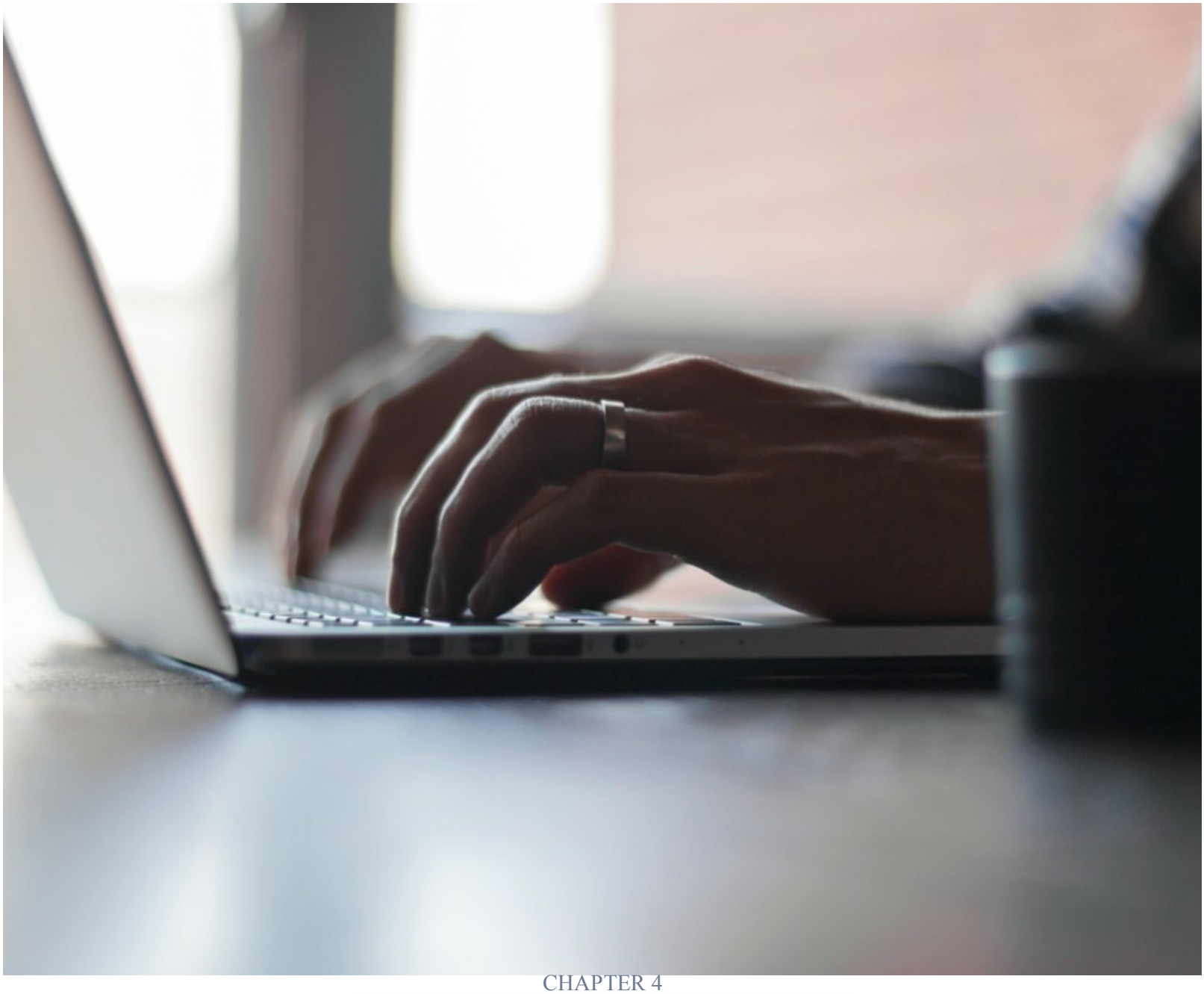
Subscribe to a few woodworking publications
I've found that it's inspiring to have a magazine chock full of ideas show up in my mailbox every few weeks. Currently I'm subscribed to Popular Woodworking and Wood Magazine, although I've also held subscriptions with ShopNotes and Fine Woodworking.

Find a Maker-Space - example: http://www.mavenmakers.com/
Facilities like this are popping up all over the country! The basic premise is that you can rent time in a fully functional woodshop where you can work on whatever you'd like. The plus is that you don't need to invest much money in tools. A number of them even offer classes!

Find a Mentor
One of the great joys of woodworking is sharing the experience with someone else! Many older woodworkers crave the opportunity to teach a beginner. See if there is someone in your sphere whom you could approach about learning the craft.

***

If you'd like to find out more about Jesse you can check out his awesome Instagram account (https://www.instagram.com/jesseschutt/) and also find him on Twitter (https://twitter.com/jesseschutt/)



CHAPTER 4

How To Write A Readme That Rocks

Developers love to share code in the form of packages, full apps, or tiny modules. Sharing is great, but one area that a lot of developers forget about is the readme file. This file is now arguably one of the most important pieces for your project and one that a lot of users don't spend any time on.

A readme first started as a guide for developers. You would access it after downloading the code, and it would show instructions related to configuration, installation, copyright, troubleshooting, and more. As popular code repositories started implementing these as the main page of the project they have now transitioned from being straight technical information to being a project home page. This shift in display means they have transitioned from straight developer docs to marketing and basic project information.

Think about the last open source project you looked at. Chances are you had a problem and

needed a solution. You found something that sounded reasonable and looked at the readme to see if it would indeed solve it. Of course, that is your primary reason for looking but you should also consider the following questions:

1. Will it solve my problem?
2. Can I trust this code?
2. Can I trust the team/developer who created it?
3. Will I be able to get help if I'm stuck?
4. Are issues or pull requests dealt with?
5. Am I willing to keep this updated if this project dies?
6. Can I build this easier myself?

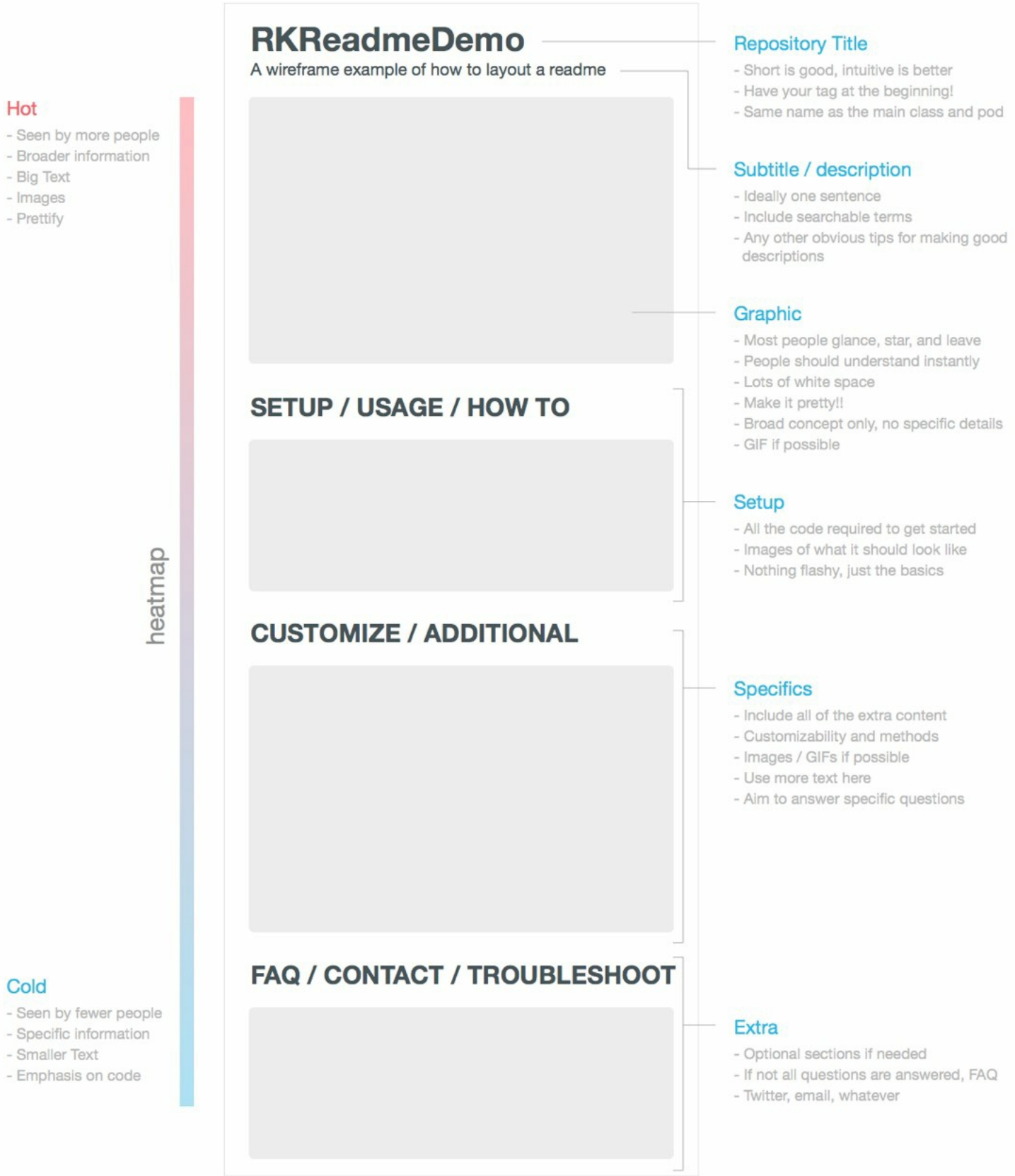
If you are not comfortable answering yes to most of these, then it's time to move on. Even if a package would work, the developer has a responsibility to instill confidence in the project because like it or not you are the one that is taking the risk by including their work.

By working through these common questions let's look at how to build a readme that instills trust, answers objections, and shows the readers why your code is worthy of being included in their project.

Readme Structure

The structure of a readme can take various forms. I've seen some cover every possible scenario, and I've seen others with one sentence: "Read the source." Of course, telling people to read the source is not ideal and covering everything is difficult, but you can find a happy medium.

Richard Kim created the following heat map showcasing how users view your readme and outline a nice structure:



He proposes an image, lead paragraph, installation, code samples, and then continuing with everything else.

I would propose changing this structure and add in an introduction paragraph between the image and the code samples. Here is the new outline:

Image

The Lead

Code Samples

Installation

Config

Everything else...

Let's go through this structure and outline what each section should accomplish and what its primary goal is.

Images and Media

Visuals are important and having one allows people sharing your project to include a picture. It also helps build confidence by showing a level of care.

"An image with the readme shows you care about the presentation," said Matt Stauffer, owner of Tighten.co. "Which makes me think it's more likely the code will have more attention paid to the details and maintenance."

If you care about presentation, then you care even more about the code. Another nice touch is to add an animated gif or a video highlighting some of the best features and highlights.

> *A looping, animated GIF will be watched over and over, scrutinized and understood. A paragraph of text will be ignored.— Taskwarrior project*

Here a few tools and resources to create these easily:

Animated Gif Tools

Licecap is a free tool that can capture an area of your desktop and save it directly to your desktop.

Droplr features screen recording and sharing.

Giphy Capture

Animated Gif Tutorials

Animated Gif Workflow by *Wes Bos*

Designing Gifs by *Invision*

Image Tools

Pablo and Sprout Social both allow you to upload an image and overlay text on it. Perfect for building images for social media sharing.

Canva is a more full featured app and offers both free and paid assets.

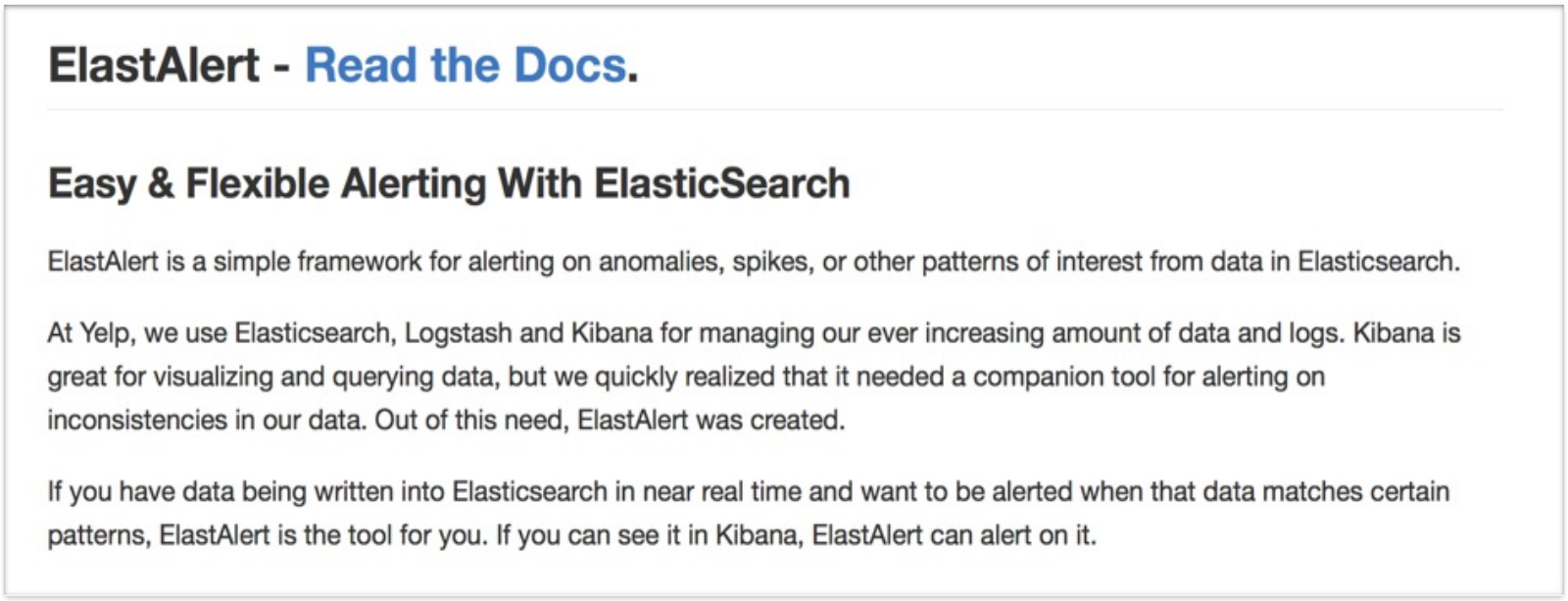
Once you have your assets ready, it is time to create your introduction.

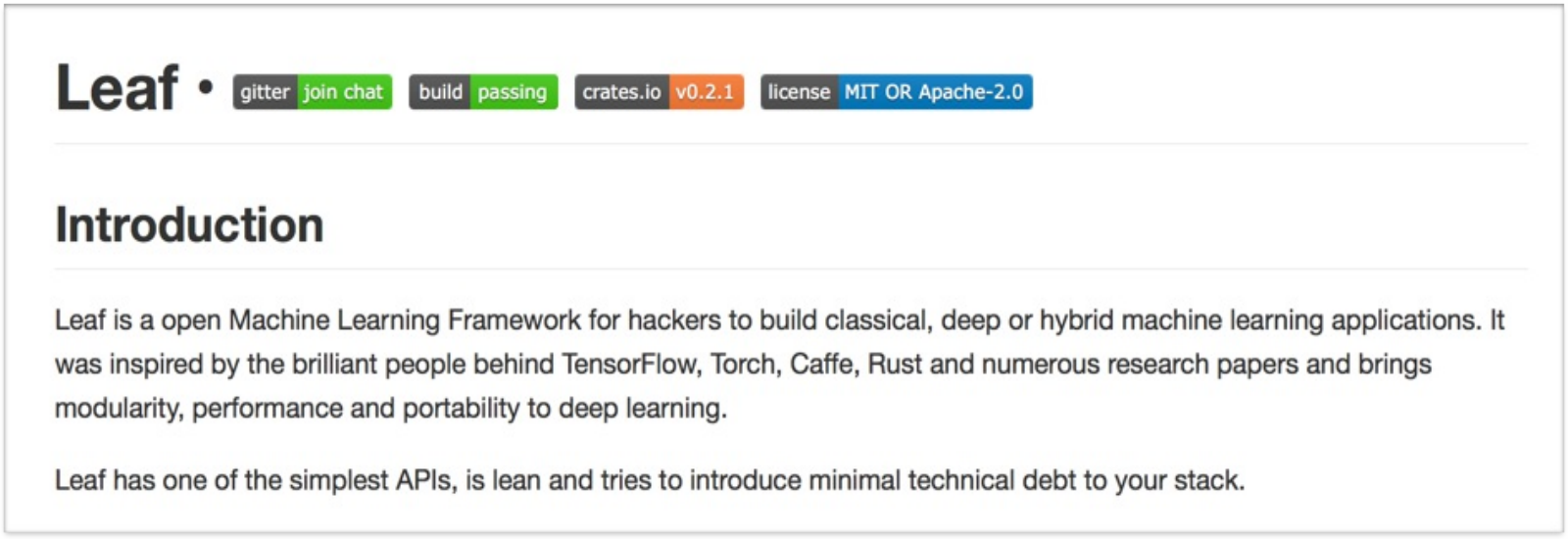
The Introduction or Lead

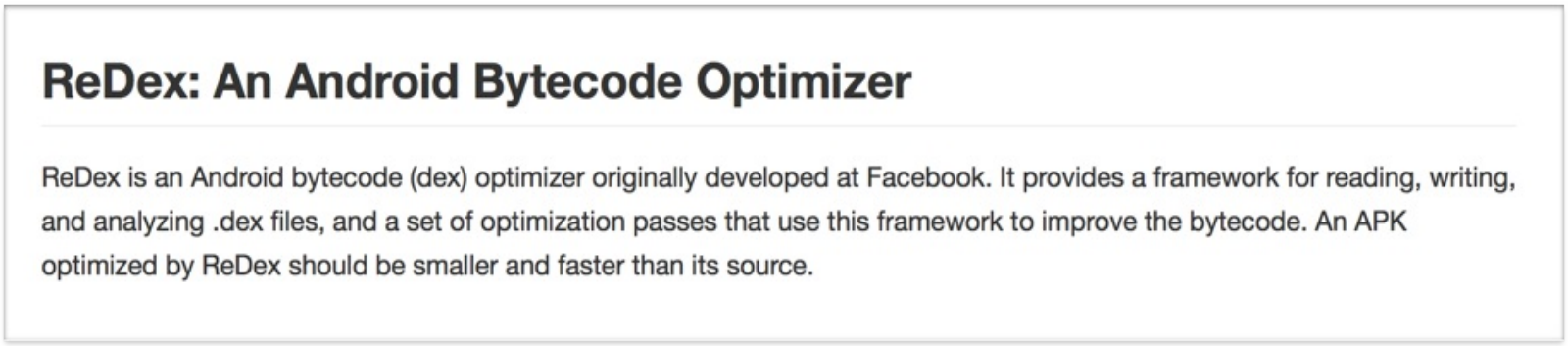
The introduction, or the “lede” in journalism, is your first chance to interest the reader and grab their attention to continue reading. It should be a few sentences that explain exactly what the purpose of the code is, why someone would want to use it, and how it works. Answer the what, why, and the how.

When you sit down to write this, it's important to remember you are the expert. You are intimate with the code, you know everything about it. Others don't. So make it clear and concise.

Here are a few examples that came directly through the trending page on Github:







Now lets move on to creating code samples.

Code Samples

After the introduction is a great place to put some select code samples. This section gives the reader a feel for how it looks and what it does, and is the first introduction into your API.

For example, if you are a building an image processing library an example to include here would be resizing and processing:

```
$image = (new Image)->resize(100, 100)->convert('greyscale');
```

Or if you are building a package to generate the total number of social shares for a URL:

```
$shares = (new SocialButler)->count($url);
```

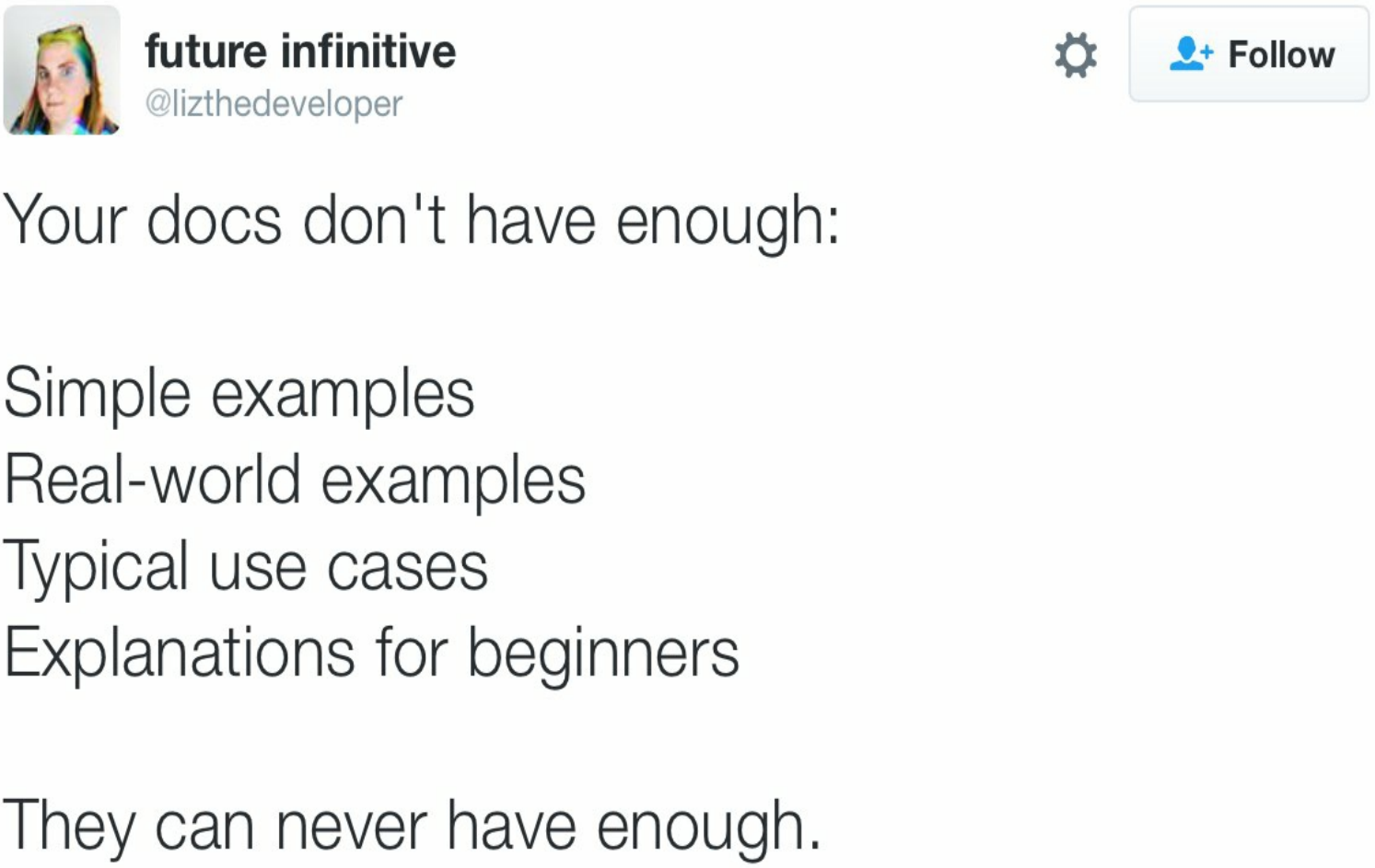
These examples are used to reinforce the trust you started building in the introduction and get the user to start thinking about how much time this is going to save them. These are not the same as documentation…that's still coming.

Configuration and Usage

Next comes the bulk of your readme. The configuration and usage outlines all the things you can do. If we continue using the fictional packages, this would be a list of all the API methods and how to use them.

It's a good reminder to put yourself in a beginner's mindset. Will they understand it? Will they be able to solve their problem with your code?

Or as @lizthedeveloper said:



Google also performed a case study on "How developers search for code," which found the most common search, representing over a third (33.5%) of the surveys, pertained to specific API, library information, or functional examples.

We aren't documenting enough of the code we put out, expecting users to either magically know how to use it or to get help elsewhere.

Copy Editing

After you've spent all this time building your documentation, you are excited to launch. However, it's time to take a step back and start editing your copy to ensure everything is clear, understandable, and free from mistakes.

Editing a readme is no different from editing a blog post or any other content. I like to walk away from it for a few hours up to a few days. That way you've created distance from your original words and come back to it fresh. Some things to look for are:

* Did you communicate clearly?

* Could sentences be improved?

* Any misspellings?

Once you've done this pass and are happy, you can utilize some online tools to improve further. I like using Grammarly and Hemingway.

Discovery

After you've polished the readme and have everything ready, how are you going to market your code? As DHH once said, "Anything worth releasing, is worth marketing."

There are many different ways of marketing your code: announce to the world on social media, write a blog post, email friends, make a forum post, etc. All these are indeed worth doing, but if you don't have a large following, it can feel as if it's falling on deaf ears.

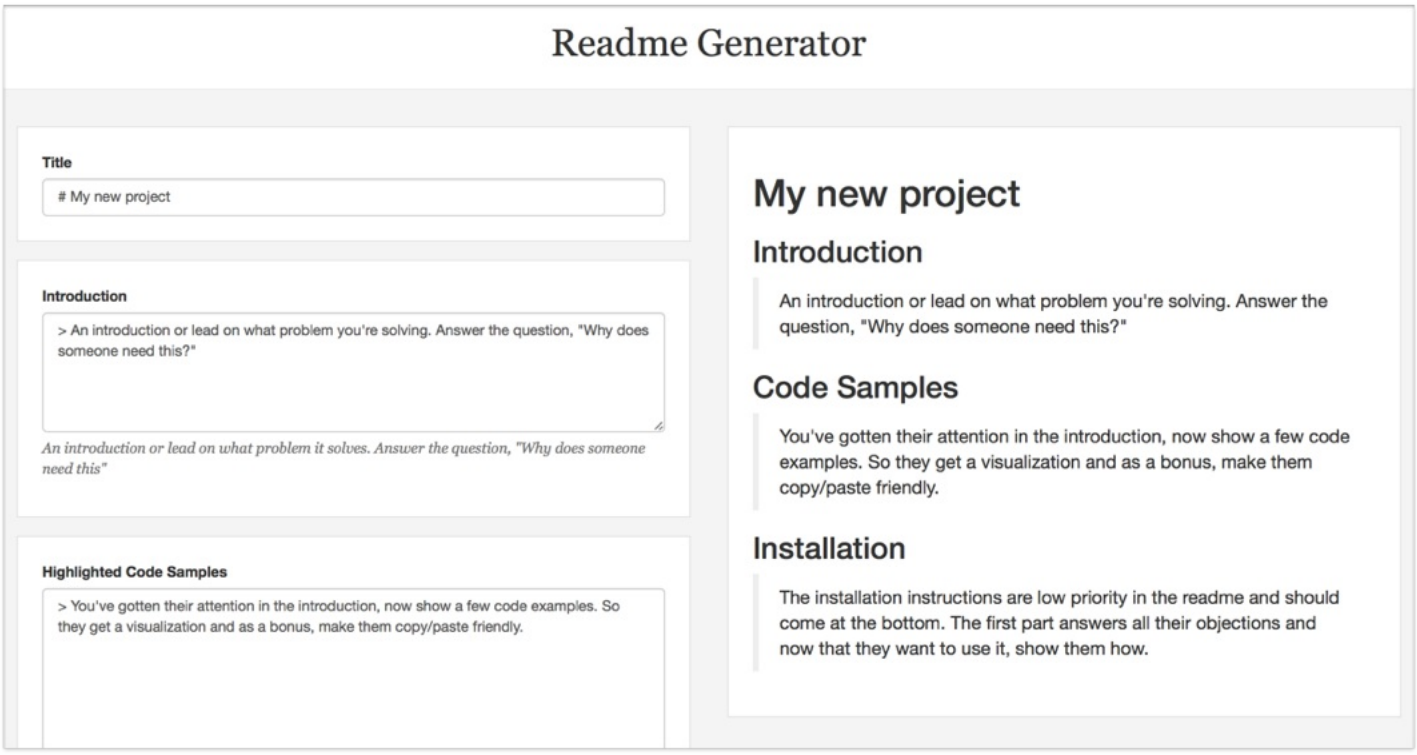
Another option is to pitch it to news sites. There are many niche news sites covering every type of programming language, and they are always looking for content. However, due to time constraints, it's difficult for journalists to spend time researching and writing about your code.

You can help them make it easy to cover your work by creating a mini press release. You already have all the resources needed in the readme so you can add the images, gif's, videos, and highlight code samples. Then create a little story around it, because everyone loves reading a story.

If you'd like more ideas on creating a press release check out this medium article on the subject and how they pitched an iOS keyboard to the media.

Readme Generator

To make creating readme's outlined here easier I teamed up with Michael Dyrynda and created an open source readme generator



Just fill out the form and you can see a live preview on the right side, then once you are happy hit "fetch markdown" to copy and paste into your projects.

With this guide and the free resource, you can take your readme to the next level.

Building The Readme Generator

By [Michael Dyrynda]

A little while ago, Eric L. Barnes reached out to me to assist him with an open source idea he had; the concept was to build a simple single page application to help people authoring projects to create a really solid README file to accompany it.

In covering so many different packages on Laravel News, the aim was to an outline a method of writing a README that really sells the package to the people wanting to use it. In his words, there are so many that are vague, unclear, and just unstructured I was thinking this could help others.

> *Basically, I feel like the beginning of the README should almost be like a press release. Sell the user on why this will make their life easier and move installation to the bottom because it's a low priority.*

What follows is a tutorial on how I built this app.

Tooling

Spending a lot of time in the Laravel Framework in my day job, I spend a lot of time with the tools that come with it. If you've ever had to use Gulp before, you know that as good as it is, it's a daunting task to write the same boilerplate every time you want to start a new project.

The main frontend build took that Laravel has available to it, is Elixir - not to be confused with the Elixir programming language. Matt Stauffer wrote a great post over on the Tighten Co blog on getting up and running with Elixir and Vue.js - even outside of a Laravel project.

Leveraging Elixir for this project meant that the entire gulpfile is just a few lines long:

```
var elixir = require('laravel-elixir');

require('laravel-elixir-vueify');

elixir(function (mix) {

    mix.sass('app.scss').browserify('app.js')

});
```

In addition to Elixir, I made use of a couple of other JavaScript utilities to put this app together:

Underscore.js - to iterate over individual README components

VueStrap - provides pure-Vue Bootstrap components without needing to add jQuery

Marked -used to render the user input into markdown format

Clipboard.js - to copy the raw markdown to the user's clipboard

Vue.js

I've been playing around with the Vue.js framework a little bit across a few projects, so it made sense to reach for this framework on this project. The Vue website has an example for a simple markdown editor.

What isn't clear from the embedded JSFiddle, however, is that the marked filter used to generate the HTML output is provided by an external package. This was a stumbling block for me initially, so something to watch out for.

This is a matter of installing the library:

```
$ npm install marked --save
```

importing it in your main app file:

```
// app.js

var marked = require('marked');
```

running gulp to rebuild your static assets, and then declaring the new filter:

```
filters: {

    marked: marked

}
```

Main app

The main application boils down to a simple JavaScript option that holds the source content of our rendered markdown. We set some default values so that the app user has some understanding of what their content might look like, as well as some pointers on what they should be writing about for each section.

```
  data: {

      markdownSource: {

          title: '# My new project',

          headingIntro: "## Introduction",

          introduction: "> An introduction or lead on what problem you're solving. Answer the question, \"Why does someone need this?\"",

          headingCodeSamples: "## Code Samples",

          codeSamples: "> You've gotten their attention in the introduction, now show a few code examples. So they get a visualization and as a bonus, make them copy/paste friendly.",

          headingInstallation: "## Installation",

          installation: "> The installation instructions are low priority in the readme and should come at the bottom. The first part answers all their objections and now that they want to use it, show them how.",

      }

      generatedMarkdown: ""

  }
```

Each of the non-heading properties on the markdownSource object are then bound to a text input in our HTML form, for example:

```
  <!-- Using Bootstrap -->

  <div class="form-group">

      <label for="intro">Introduction</label>

      <textarea class="form-control" v-model="markdownSource.introduction" debounce="100" rows="5"></textarea>

      <span id="helpBlock-introduction" class="help-block">An introduction or lead on what problem it solves. Answer the question, "Why does someone need this?"</span>
```

```
  </div>
```

The introduction form snippet is bound to the introduction property using the v-modelattribute with a debounce value of 100 (ms). The debounce parameter allows you to set a minimum delay after each keystroke before the input's value is synced to the model.

We repeat this for each of the other fields declared in the markdownSource object - title, codeSamples, and installation.

Rendering Markdown

As we have already setup our filter using the marked library, Vue makes this process really simple:

```
  <div id="markdown-preview">

      {{{ markdownSource.title | marked }}}


      {{{ markdownSource.headingIntro | marked }}}

      {{{ markdownSource.introduction | marked }}}


      {{{ markdownSource.headingCodeSamples | marked }}}

      {{{ markdownSource.codeSamples | marked }}}


      {{{ markdownSource.headingInstallation | marked }}}

      {{{ markdownSource.installation | marked }}}

  </div>
```

That's it.

Vue handles binding changes to the model from our input, as well as filtering the markdown into HTML for the preview.

Ordinarily, we would use double braces - {{ dataToOutput }} - however, because we are rendering HTML, we don't want it to be escaped, so we use the tripe brace variation.

Fetching Markdown

Once our user finishes writing their README, they need some way to get the raw markdown in order to drop into their project's README.md. This is where we leverage the Underscore, VueStrap, and Clipboard.js libraries.

Clicking on the Fetch Markdown button triggers a few actions to take place.

First of all, we ensure that the title heading is preceded with a #, which will be translated into a HTML <h1> tag. As this is the only heading field that the user can modify via our app, we need to ensure it's correctly formatted.

```
if (this.markdownSource.title.substring(0, 1) !== '#') {

    this.markdownSource.title = '# ' . this.markdownSource.title;

}
```

Next, using the Underscore library, we iterate over the markdownSource object and populate the generatedMarkdown property on our data object.

```
this.generatedMarkdown = _.map(this.markdownSource, function (text) {

    return text;

});
```

Lastly, we set the showMarkdownModal property on our data object to true, which VueStrap's modal component is configured to display on.

```
<modal :show.sync="showMarkdownModal" effect="fade" title="Your README that rocks" :large=true>
```

The modal is displayed with a text area containing the raw markdown, bound to the generatedMarkdown property and filtered to preserve new lines:

```
  <textarea class="form-control" id="generated-markdown" rows="20" v-model="generatedMarkdown | nl2br">
</textarea>
```

The nl2br filter is a custom filter that was created to join the generatedMarkdown array into a string, separated by two new line (\n) characters.

The modal window has a Copy button, with the generated markdown textarea as it's target:

```
  <button type="button" class="btn btn-primary clipper" data-clipboard-target="#generated-markdown"
@click="outputCopied = true">Copy</button>
```

The last piece here is setting outputCopied = true. This is used to provide feedback to the user that their content was copied to the clipboard. In the event the copy failed, such as in Safari and unsupported versions of other browsers, a message is displayed to tell the user to copy the content as normal (Cmd/Ctrl + C), as the content is highlighted and focussed for them automatically.

Conclusion

I've covered a fair bit in this article, so here's a quick recap of what I went over:

Generate rendered HTML from Markdown with ease, by using the Marked library as a Vue filter.

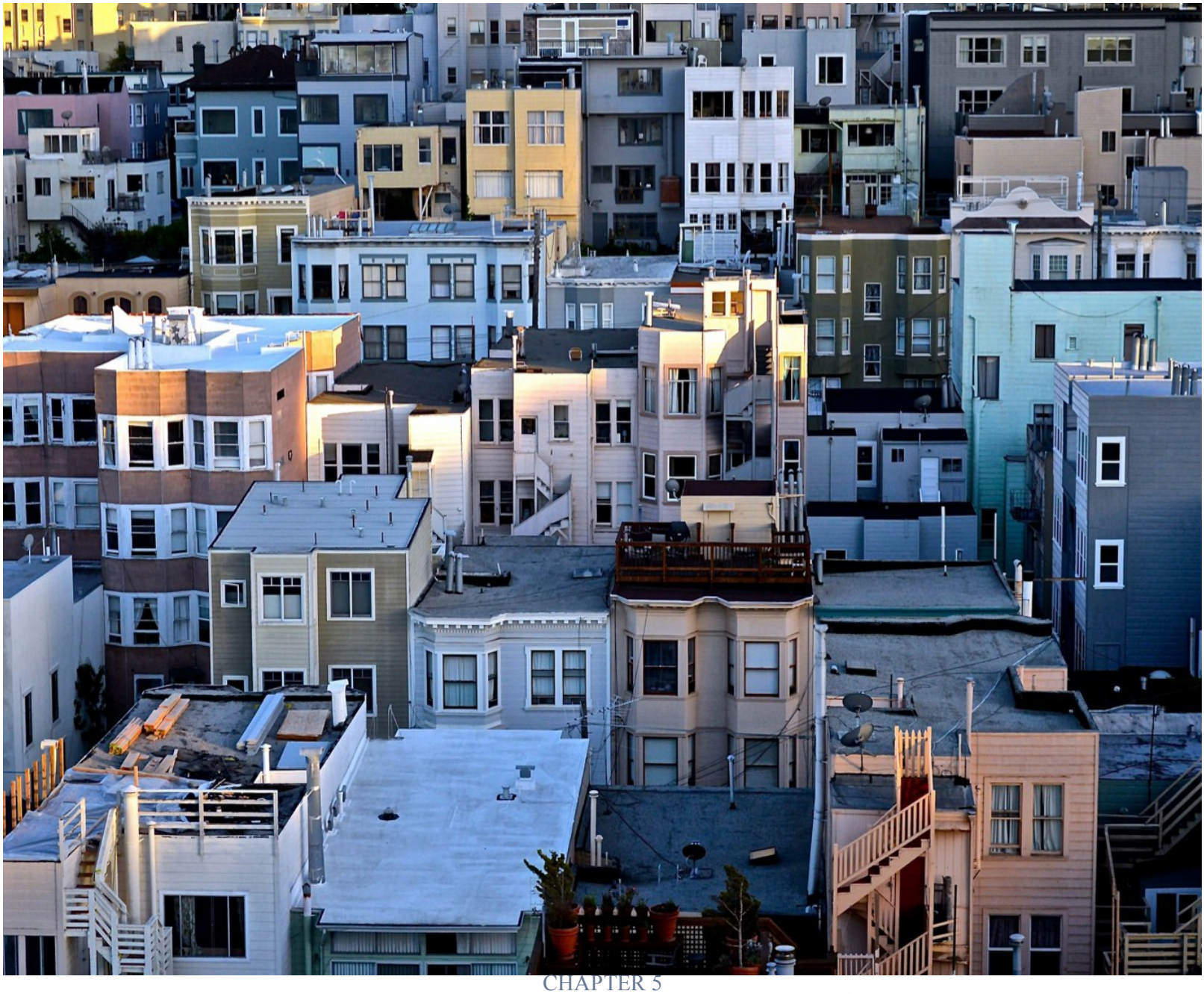
Binding user input from form fields to Vue's model, so that we can render filtered markdown with ease

If you're using Vue, and still want some Bootstrap JavaScript components, VueStrap is an excellent tool to use, without needing to bring in jQuery as well.

Render raw markdown in a textarea by iterating over our markdownSource object's children and letting users easily copy the output using Clipboard.js

If you have any questions about the project, feel free to contact me on Twitter. If you'd like to contribute, feel free to open an issue or pull request on GitHub.

I hope that you find this simple application useful in writing a README That Rocks for your next project! I'd love to hear from you if this project has been helpful to you.



CHAPTER 5

Community Corner

So much is created with Laravel every month that it's impossible to cover every item. In this community corner I wanted to highlight some items released in the last month that I believe are worth checking out.

Code Smart *by Dayle Rees*

Dayle released his latest ebook for learning Laravel. If you are interested in it you can use the coupon code LARAVELNEWS for 10% off.

Laravel Moderation

This package setups up a simple moderation system so you can approve or reject items. I thought this was great because I recently had to do this manually on the links.laravel-news area.

San Diego Laravel Meetup

A recorded live stream of their latest meetup.

Laravel 5.1 Cheat Sheet

A cheat sheet you can use to quickly find Laravel features.

Laravel Test Tools

Create Laravel tests directly through a Chrome extension