Technology at Vegan Hacktivists Pages Pages

In this book, we cover the technology stack(s) used at VH, the reasons for why, and our approach to development.

Our Tech Stack

VH has many active projects, and for the sake of maintenance and knowledge transfer, we'd like all projects to share a common stack. While the critical components of our projects are mostly the same, advances in technology have caused projects to diverge a bit over time. This page serves as a living document meant to help make sense of those divergences.

Main stack(s)

Shared Technologies

Hosting

With the exception of VeganHacktivists.org, **all of our projects are hosted on a DigitalOcean server managed by Laravel Forge.**

The server is backed up by Digital Ocean once per week.

Local Development

Laravel has a wonderful Docker setup called Laravel Sail, which should be the *only* choice for running our projects locally.

A number of our projects were built before this was available, so we are all over the place with this one. Our older projects collectively use custom Docker setups, Gitpod and Laravel Homestead, and some people are even using Laravel Valet to develop locally. **All of these methods should be considered legacy, and if time permits, we should be moving over to Laravel Sail.**

Admin Panel

With the exception of Vegan Bootcamp, all projects that require an admin panel should be using Backpack. To set this up, follow the guide for setting it up.

CSS

All new projects should use the Tailwind CSS framework. A number of our older projects use Bootstrap for legacy reasons, but this should not be considered for future projects.

JavaScript Build Tool

Vite is currently the recommended build tool for Laravel projects and is set up by default. Most of our projects were started before this was the case, so they use Laravel Mix which is a wrapper around Webpack. It is recommended to switch to Vite if possible.

Package Management

We use Yarn for managing packages in our projects. Originally, this was the default choice for Laravel, but as of Laravel 8 or so, it has switched to NPM. To keep things consistent, we will continue to use Yarn.

Code Formatting

This is another fragmented part of the stack, but as of now, the recommendation is to use Laravel Pint for PHP, and Prettier for everything else. For the latter, please refer to our base Prettier config.

Some projects are using Prettier or PHP-CS-Fixer to format PHP code, and others are using nothing at all. It is recommended to set up Pint for PHP and Prettier for everything else if possible.

Laravel / Inertia.js / TypeScript / React

As of now, this should be considered the go-to stack for most new projects.

Why Choose This?

Typically, building a single-page app requires you to build an API, which introduces a lot of overhead to the development process. However, with Interia, all of that overhead disappears.

With Inertia, you are able to build a single-page app without the API. In essence, you build your back-end the same way you normally would with a plain server-rendered front-end, but your data gets sent to a React app instead.

There are several reasons for choosing React for the front-end. React is easily the most popular front-end framework, which makes it easy to find people who can work on VH projects. It also has an extremely rich ecosystem as a result, which makes building complex apps much easier.

Other technologies we've used in the past (e.g. Alpine and Livewire) made things a bit more difficult because of the smaller ecosystem. Whenever we needed more complex components, such as comboboxes, we usually had to reach for clunky solutions or implement them ourselves. By using React, we don't have to do that anymore.

As for TypeScript, the reason for it is pretty simple: a type system gives us more confidence in our code, especially during things like refactors. Static typing also lends itself to more powerful dev tools and helps make certain functionality even better (e.g. autocompletion in your text editor).

Why *Not* Choose This?

Under very specific circumstances, it may not make sense to choose this tech stack. Mainly, it has to do with the size and complexity of the project.

If the project is complex enough that it needs some sort of server-side logic (i.e. Laravel), but is simple enough that the front-end wouldn't require or even benefit from even a moderate amount of JavaScript, then it probably isn't worth using this stack, since it would lead to unnecessary complexity and a much larger JavaScript bundle.

Notes

- To get started with a project with this stack, follow the guide for starting a new project.

Relevant Projects

- PitchFTA
- Open Sanctuary
- Watch Dominion
- Phauna Grant Applications project

Laravel / Alpine.js / Livewire

This stack should be considered legacy, but it is still permissable to use it under certain circumstances.

Why Choose This?

The main reason to use this stack would be to enhance apps built using an even more legacy tech stack. This is because it's much easier to introduce Alpine and Livewire in small bits, whereas Inertia would require entire pages to be rewritten, and they'd also come with JavaScript bundles that include all of React.

A good candidate for this would be something like Vegan Bootcamp, which was built before these libraries existed. If a project has had unanticipated changes in requirements that now demand more interactivity, it would probably make sense to introduce Alpine and/or Livewire to the project instead of rewriting everything using Inertia/React.

Why *Not* Choose This?

The main reason is the smaller ecosystem/community. While the technologies are great at what they do, there are far fewer off-the-shelf solutions for common UI problems, thus making it more likely that you'll be writing a lot of custom code. The solutions that *do* exist are much less likely to be maintained for a long time, also due to the small community.

Relevant Projects

- Vegan Linguists
- Activist Hub
- Today For Animals
- My Daily Dozen

Laravel / Intercooler.js

This is an ancient tech stack used for only one project: Vegan Bootcamp. The combination of Alpine and Livewire are far more powerful, and in the case of the latter, much more deeply integrated into Laravel, rendering Intercooler useless for our projects.

Relevant Projects

- Vegan Bootcamp

Plain Laravel

There's not much to say about this. It's just Laravel with pages rendered by the server using Blade templates.

Why Choose This?

You may look into this if you are building a very simple website that is *just* complex enough to require server-side logic.

Why *Not* Choose This?

If the project is going to require even a moderate amount of client-side interactivity, you are likely going to be served far better with the Inertia/React stack.

Relevant Projects

- FAST Community (note: also uses Alpine)
- Meat the Victims
- Vegan Activism
- 5 Minute 5 Vegans

Plain PHP

This is even simpler than using Laravel. It's PHP without the framework.

Why Choose This?

There's not much of a reason to choose this, especially for future projects we're likely to build. However, it's permissible if an application is so simple that it only requires a little bit of server-side code, maybe 1-2 pages, and no database.

Why *Not* Choose This?

If you have a multi-route website that requires more than a couple hundred lines of server-side logic or anything moderately complex (e.g. authentication, database interaction, etc.), then Laravel is likely going to be the better choice.

Relevant Projects

- Animal Rights Map

Plain React

Short of plain HTML, this is the simplest stack of all.

Why Choose This?

This should of course only be used for projects that require no server component, but require rich interactivity in the client.

Why *Not* Choose This?

If you need server-side logic, of course, don't choose this tack.

If you can get by easily without access to React or its ecosystem, it might be best to go with plain HTML.

Relevant Projects

- Wild Animal Suffering

Recommended Libraries

Radix UI

This is a React library that makes it very easy to built complex UI components that adhere to accessibility standards.

Thanks to Stephan for the recommendation!

Laravel Query Builder

This is a Laravel library that reduces the need for boilerplate when implementing common functionality related to database querying, such as sorting and filtering.

Thanks to Joaquín for the recommendation!

Laravel TypeScript

This library parses your Laravel models and creates TypeScript interfaces out of them, which eliminates a lot of work in Inertia/TypeScript/React projects.

Thanks to Joaquín for the recommendation!

VeganHacktivists.org

[To be continued by Tobi or someone else from Team Avocado]

Data Projects

[To be written by Stephanie or someone else from Team Strawberry]

Why PHP/Laravel?

Note: This page will be written in the first person, from the perspective of Vegan Hacktivists's former Director of Engineering, Gerard O'Neill.

TL;DR

- We have been using Laravel for four years and are deeply entrenched in the ecosystem.
- Introducing more technology would lead to more maintenance headaches and higher operation costs.
- Projects would not be built as quickly with other technologies.
- PHP and Laravel are actually really good if you give them a chance.

Our tech stack is a topic of conversation more often than I like it to be, and far more often than I ever expected it would be. I have always viewed programming languages merely as tools, especially in the context of web development, where the differences between most of the popular ones (PHP, Python, Ruby, JavaScript, etc.) are so minor. Sometimes, a volunteer will express their dissatisfaction with our tech stack and propose that we use different technologies to build our projects. Because of this, I would like to do a deep dive into why we use Laravel for our projects, and why we will not be using other frameworks.

The Origin Story

First, it's important to know the history of VH in order to understand the context surrounding our tech stack. The organization started as a very tiny group of Redditors, led by David. David is not a software engineer by trade, but had some experience building websites, mostly using PHP (WordPress, PHP Fusion, some basic custom sites, etc.). The word "deploy" was not in his vocabulary. To update a website, hosted on the shared hosting provider HostGator, he would FTP into and upload files directly onto the server.

This is how the first VH projects were built. David was somehow able to rally up a few programmers willing to work on some toy projects in this manner. Within a few months, Git was introduced, but the "deployment" process remained the same. This obviously couldn't scale for so many reasons: poor security, the need to build assets locally in production mode before uploading files via FTP, etc.

I joined VH several months after it was started. By that time, two PHP projects had already been built without a framework. I was assigned to help with the third project, which was the first one built using Laravel. This was mostly new for me, since I had experience with PHP, but I had never built a project using Laravel. At that point, I had spent the last three years working exclusively with JavaScript, and hadn't so much as looked at PHP since my previous job.

Despite HostGator not being able to support very many different technologies, the original plan was for VH to build out one new project per month, built with a different technology each time. I probably don't need to explain why this is a terrible idea, but just in case it's not obvious: **This would've been a maintenance nightmare.** I couldn't handle the idea of cranking out twelve hastily-coded projects per year, all needing to be maintained in their own unique ways, with an ever-changing team of volunteers.

I had a chat with David and had no trouble convincing him that this was not a good idea. Thus, we made the decision to stick with Laravel. It's the most popular PHP framework, we had already shipped with it before, and everyone on the team at the time seemed pleased with it. We also made the decision not to try to release a new project every single month.

Fast-Forwarding to Today

Long gone are the days of FTP uploads to a shared HostGator server. We have over a dozen PHP projects hosted on a Digital Ocean server managed by the Laravel Forge service. Deploying is as simple as updating the main branch. Now that we know the history of VH, as well as the current state of things, let's go over some of the reasons that we will be sticking with the current stack for all future projects.

It's Less Maintenance

We have built up years of experience managing Laravel projects, and Laravel Forge makes it virtually unnecessary to have people with DevOps experience to help with their upkeep.

If we introduce a new stack into the organization, we will not benefit at all from the experience we've gained up to this point. If we built an app with a different technology, we'd need to host it somewhere else. We'd need to make sure that we always have at least one person who knows how that project works and how to maintain it, because if all volunteers with that knowledge leave, there wouldn't be anybody who would know what to do if something breaks. This is especially important for VH, which is run almost exclusively by volunteers, none of whom we expect to be available when things go wrong.

If this isn't convincing enough, I highly recommend this talk about choosing "boring technology."

It's Faster

This will sting for some and be counterintuitive for others, but it's true. Building our projects in Laravel results in shipping faster. We do have volunteers that don't have experience with Laravel or even PHP, but this drawback is vastly outweighed by the speed gained by sticking with Laravel.

The first thing to note is that Laravel, unlike virtually all JavaScript frameworks, is opinionated with many batteries included. Listed below are a bunch of things that are already been built into the framework or offered as a *first-party package*, reducing the need for debate amongst teammates about which libraries to use, as well as eliminating boilerplate code and time spent setting things up.

1. Testing framework, from unit tests all the way up to headless browser testing
2. ORM
3. Database migration functionality
4. Drivers for various data stores, such as Redis, MySQL, etc.
5. Email functionality (both the templating and the sending of emails)
6. Notification functionality (both via email and stored in the database)
7. Localization functionality
8. OAuth functionality (both client and server)
9. Rate limiting
10. Code formatting
11. Full-text search
12. Job running (async/queued jobs as well as scheduled jobs)
13. Asset building (using Vite)

There's a lot more, but they're less relevant to the needs of VH. In addition to all of that, the Laravel ecosystem also provides us with the following, all of which are also built by the Laravel core team:

- Laravel Sail - A standardized Docker environment which sets up everything you could ever need to develop a Laravel project.
- Laravel Jetstream - A starter project that gives you a fully-built authentication system (including things like password reset emails and 2FA), profile editing, session management, teams functionality (optional), and an API for all of the above. It also has things like Tailwind CSS set up already.
- Laravel Breeze - A more lightweight version of Jetstream with fewer features out of the box.
- Laravel Forge - Mentioned above, the service that manages our Digital Ocean server and provides us with the ability to manage tons of stuff: scheduled jobs, daemons, PHP versions, configuration files, SSH keys, environment variables, Redis queues, SSL certificates, automatic deployments, failed deployment notifications, *and more*.

- Laravel Envoyer - A service for handling zero-downtime deployments, as well as many other things (heartbeat checks, deployment hooks, even more notification possibilities, etc.)

Last, but not least, we were also given a free lifetime license to use Backpack, an extremely powerful third-party admin panel for Laravel.

All of the things listed above are technologies we actively use. If we were to build a project using a JavaScript framework, we would not be able to use any of it. In order to get the same functionality, a non-trivial amount of time would need to be spent building it from scratch, or at best, writing boilerplate code in order to connect many third-party tools together. This doesn't even take into account the time spent figuring out which technologies to use in the first place, nor does it take into account the added DevOps requirements we'd likely have with other technologies.

Something also worth noting is that using the same technology for each of our projects allows for much more code reuse. If a similar feature has already been built in a previous project, it is typically very easy to replicate that functionality in a newer project, either with a simple copy/paste or by taking inspiration from the existing code.

In short, Laravel helps us ship faster.

For the sake of completeness, I should note that certain frameworks such as Ruby on Rails, Django, Phoenix, etc. are also opinionated, but for some reason, virtually all complaints about PHP and Laravel come from those wanting to use a JavaScript framework, hence the focus I place on JavaScript in this section.

Addressing Common Concerns

In case everything up to this point hasn't been convincing enough, let's go over some issues that have been raised over the course of VH history.

"I Don't Have PHP Experience"

This is probably the most valid objection to using PHP. As much as I believe that web programming languages are very similar, and that a good developer should be able to get up to speed in a new technology quickly, it still does take some time and effort. This is obviously a bigger ask for a volunteer than for a full-time employee.

However, as stated above, Laravel makes us faster. If we were to use another technology, we would simply be trading time spent learning for time spent debating/making decisions, writing boilerplate code, and maintaining a bigger, more complex set of technologies. It's simply not a good trade.

"I Don't Like PHP"

This is probably the most common objection that we've gotten, though in my personal experience, this statement is actually an alias for the above one. To put it more bluntly, the vast majority of people who claim to dislike PHP are those who have never (or almost never) used it, thus don't have enough experience with it to have an informed opinion. Even if that isn't true for someone, it's not enough of a reason for us to fragment and complicate our stack.

"PHP Can't Do X Well"

There are certainly projects where PHP would make less sense to use. If we were building a project with lots of real-time functionality that made heavy use of WebSockets, then it *might* make sense to build it using a different technology. With that being said, we would still need to explore using PHP as an option before making that decision. In order for us to use a different technology, the benefits would need to *significantly* outweigh the drawbacks. Again, I highly recommend reading Choose Boring Technology for more perspective.

In any case, none of our projects have ever had requirements that were even remotely difficult to satisfy with Laravel, so this is a non-issue.

"PHP is Dead" / "Nobody Uses PHP"

This is something I've been hearing for at least the last ten years, and it's about as true today as it was back then. Not only is Laravel an extremely popular framework, but technologies like WordPress, Drupal, etc. are used for so many websites that PHP powers almost 80% of the web.

The language itself is constantly improving over time, not only by adding more modern functionality, but also introducing significant performance improvements.

Here's a quick list of companies/websites you've probably heard of which use PHP:

- Facebook
- Wikimedia
- Mailchimp
- Etsy
- Slack
- HelloFresh
- Tumblr
- 9GAG
- Upwork
- Skillshare
- DeviantArt

- Tailwind UI

PHP is not dead, and it isn't going anywhere anytime soon.

Moving Forward

I hope this article explains why we use the technologies we use. I know that regardless of how much it makes sense for us to keep using Laravel, it will cause unhappiness for some people. However, it's important to realize that this is true of any tech stack.

After all of this, you may be of the opinion that I'm simply a Laravel fanboy or a PHP apologist. While those things may well be true, it's worth noting that I am a pragmatist above all. If the first several VH projects were built in Ruby on Rails, Django, or Express, I would've advocated for us to continue building apps with that technology, and we'd probably be using one of those instead of Laravel today.

Thanks for taking the time to read this post. Let's keep coding for the animals!

The Deployment Process

Credential Sharing

At the moment, we use Keybase for sharing credentials, secret files, etc. To get set up, you will need to download and install the appropriate app for your operating system.

On keybase, we currently have four "teams" set up on Keybase:

1. General (`veganhacktivists`) - For credentials that can be used by all VH members.
2. Dev (`veganhacktivists.dev`) - For credentials relevant to all VH developers.
3. DevOps (`veganhacktivists.devops`) - For credentials that only the DevOps team should need or have access to.
4. Admin (`veganhacktivists.admin`) - For credentials that should only be accessible by the directors and team leaders.

If you need access to one or more of the teams, contact your team leader and let them know. If you *are* a team leader, contact the director you report to!

If you are part of all the teams you should be part of, but don't see the credentials you're looking for, let us know and we'll get that fixed. There is always the possibility that the credentials are too restricted.

Adding New Team Members

When adding a new team member to a Keybase team, please make sure they are being added to the appropriate team(s) for their role. Also, please use the following guideline for what their role is in the Keybase team:

1. Admin - This should be reserved for directors, team leaders, and any full-time members of VH.
2. Writer - This should be reserved for the DevOps team or anyone else who is likely to be adding, updating, or removing credentials frequently.
3. Reader - This should be the default role for VH members unless they fall into one of the above categories.