2. Understand how Vue component instances work

Vue has a very clever design to improve performance and reduce its memory footprint.

While not necessary, understanding how this works under the hood will only help you as you build more and more Vue components.

Besides, it’s really interesting!

In this brief but very informative article by Joshua Bemenderfer, learn how Vue creates component instances: Understanding Vue.js Component Instancing.

3. Force Vue to re-render — the right way

In 99% of cases where something doesn’t rerender properly, it’s a reactivity problem.

So if you’re new to Vue, you *definitely* need to learn as much as you can about reactivity. I see it as being one of the biggest sticking points for new developers.

However, sometimes you need a sledgehammer to get things done and ship your code. Unfortunately, deadlines don’t move themselves.

And *sometimes*, forcing a component to rerender is actually the best way to do it (but very very rarely).

By far my most popular article, I've written about the proper way to rerender a component.

4. Vue doesn't handle multiple root nodes — yet

Not all components make sense to have a single root node.

For example, if you're rendering a list of items, it could make way more sense to simple return the list of nodes as an array. Why unnecessarily wrap it in a `ol` or `ul` tag?

This is called a *fragment*.

Currently Vue doesn't support fragments, although there will be support for them in Vue 3.

It's something that React has had for awhile now, but it took a rewrite of the rendering system in order for them to implement this. Vue is in the same situation.

However, you can use functional components to get around this issue while we wait for Vue 3.0 to be released. You can read more about that in Can A Vue Template Have Multiple Root Nodes (Fragments)? by Anthony Gore.

5. Validate your forms the easy way — using Vuelidate

At one of my previous jobs, I'm pretty sure my official job description was:

> *Builds datatables and forms*

It felt like all I was doing every day was building form after form after form.

But it makes sense. Forms are the main way that we get input from the user, and they are absolutely crucial to our applications working well. So we end up writing lots of them.

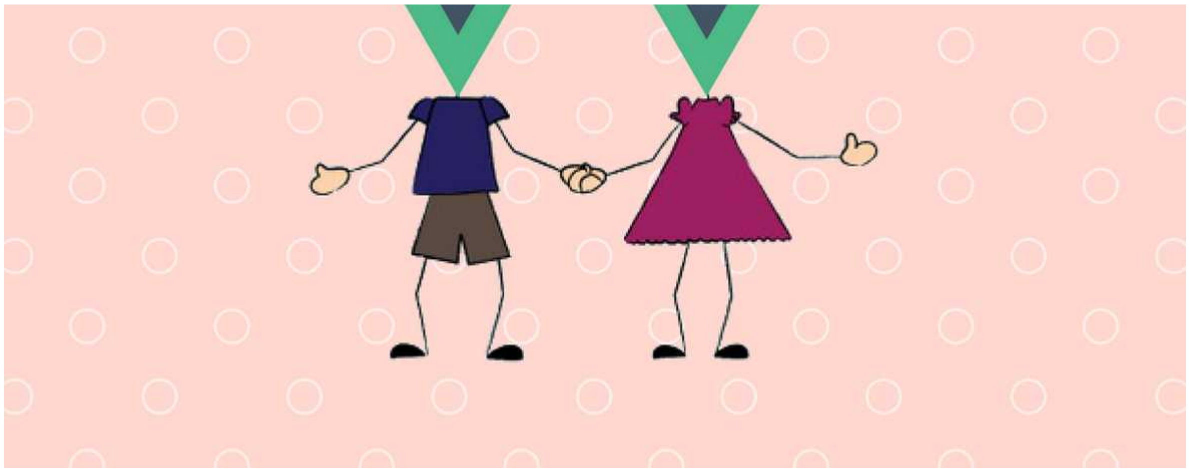
However, forms are also really tricky to build. On the surface it seems like they should be fairly straightforward to write. But as you start adding validation rules and other logic, it can quickly turn into a nightmare.

This is where Vuelidate comes in.

It's a library that makes it super easy to add custom validation, and does all the heavy lifting for you.

Learn how to setup Vuelidate by reading Simple Vue.js Form Validation with Vuelidate by Dobromir Hristov.

6. Build components that play nicely with each other



The absolutely *worst* feeling is realizing that you built your component the wrong way, and now you have refactor it completely.

You don't want to overengineer your code, but many times things like this can avoided from the start.

Kevin Ball has written an article outlining several different things to keep in mind as you write your components to keep them playing nice with others.

Check out his article: How To Build Vue Components That Play Nice

7. Don't write one-off transitions — make them reusable

Transitions are a really cool feature in Vue. If you haven't had a chance to check them out, they're a really easy way to add nice animations into your app.

But you don't want to keep re-writing the same thing over and over again, do you?

In a great article, Cristi Jora shows us how we can write a component to make our transitions reusable. It also demonstrates some great concepts for how we can make our code more reusable, and can apply to other parts of your app as well.

Check out the article here: Creating Reusable Transitions in Vue

8. Learn how to use Axios for data fetching

Almost every app needs to fetch or post data.

The most popular library to help us do that these days is Axios. It is really configurable, and makes working with external data *so* much easier.

You can just use the browser's built-in `fetch` most of the time. But you'll probably end up writing a wrapper for it anyways, to make common cases more convenient to deal with.

Might as well just start off on the right foot and use axios from the beginning then!

Learn how to integrate `axios` into your Vue app by reading Vue.js REST API Consumption with Axios from Joshua Bemenderfer.

9. Use vue-router to handle client-side routing

If you're doing client-side routing, hand-rolling your own solution isn't that difficult.

It's actually pretty simple to match routes, then swap between different components.

But just using `vue-router` is *so much easier*.

It's also an official Vue package, so you know it will always work really well with Vue.

And once you start dealing with:

- queries
- route params
- nested routes
- dynamic route matching
- transitions

...which you probably will, writing your own solution gets to be very cumbersome.

Instead, just check out this guide from Ed Zynda on Getting Started With Vue Router

10. Create filters to reuse formatting

Formatting data to display on screen can get annoying.

If you're dealing with lots of numbers, percentages, dates, currencies, names, or anything else like that, you'll likely have functions that format that data for you.

Vue comes with this great feature called *filters*, which was inspired by Angular.

They let you easily encapsulate these formatting functions and use them in your template with a really clean syntax.

But don't just take my word for it.

Check out this very detailed article from Rachid Laasri, which has tons of examples on how to write your own filters: How to Create Filters in Vue.js with Examples

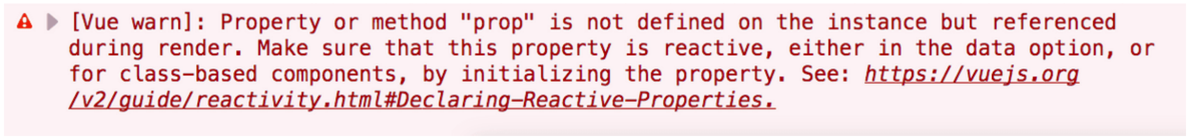
11. Make sure to avoid annoying errors and warnings

As web development has gotten more complex, so have our tools. These days we have linters, editors, type checkers, and all sorts of things that save us time by telling us our mistakes (almost) as soon as we make them.

Vue also has really good warning and error messages, but if you keep getting them it can be annoying.

Why not just avoid them altogether?

One of the most common warnings I got when I was first learning Vue was this one:



```
[Vue warn]: Property or method "prop" is not defined on the instance but referenced
during render. Make sure that this property is reactive, either in the data option, or
for class-based components, by initializing the property. See: https://vuejs.org
/v2/guide/reactivity.html#Declaring-Reactive-Properties.
```

Luckily, I wrote an entire article on what causes this, and more importantly, how to avoid it altogether!

12. Don’t be afraid of JSX — it’s extremely powerful

A lot of people are intimidated by JSX.

I get it.

It has a weird syntax, and it can be difficult to wrap your head around how it's used.

But sometimes — especially when writing higher-level reusable components — a template just doesn't cut it. You need to take advantage of the full power of the `render` method.

And JSX is one of the easiest ways of doing that.

Samuel Oloruntoba has written a great introduction to JSX and why exactly it is great that Vue has support for it: Using JSX with Vue and Why You Should Care

13. Figure out how to react to mouse hover

In CSS it's pretty easy to change things on `hover`. We just use the `:hover` psuedo-class:

```css
.item {
  background: blue;
}

.item:hover {
  background: green;
}
```

In Vue it gets a little trickier, because we don't have this functionality built in.

We have to implement most of this ourselves.

But don't worry, it's not *that* much work.

I've written an in-depth article on using hover in Vue. It covers a lot of different things you'll want to know:

- How to implement a **hover effect** in Vue
- How to **show an element** on mouseover
- How to dynamically **update classes** with a mouseover
- How to do this even on **custom Vue components**

Check out the article: How to Implement a Mouseover or Hover in Vue

14. Add v-model support to custom components

As web developers, our jobs revolve around getting data from inputs.

Vue gives us `v-model`, which is some syntactic sugar that creates a two-way data-binding for us. This is great for inputs, as it simplifies working with them a lot.

But did you know you can add `v-model` support to your own components?

Joshua Bemenderfer shows us how this can be done in Adding v-model Support to Custom Vue.js Components.

15. Fix "this is undefined" error

Perhaps one of the most common errors to run into is this one.

I used to run into this one *all* of the time. But now I know exactly what I was doing wrong. It has do with the type of function you're using, and how you're using it.

But I won't get into that here.

Learn how to fix the "this is undefined" error, and get on with your life!

16. Use an off the shelf CSS framework

Getting all of your CSS *just right* can take an extraordinary amount of time.

My suggestion is to just use a CSS framework, where most of the work is already done for you.

All of the styling, colours, drop shadows, and aesthetic elements are already worked out. No need to learn graphic design! On top of that, they all come with tons of CSS styles to help you with layout, forms, and other common elements like buttons, popups, alert boxes, and so much more.

The best part is the variety.

There are tons of great ones to pick from:

- Tailwind
- Bootstrap
- Bulma
- Foundation

And don't waste any time trying to figure out how to integrate them into your app. Dave Berning has written a great article to help get you started: Integrating and Using CSS Frameworks with Vue.js.

17. Watching nested data in Vue

Watchers are a really great feature in Vue. They make adding side-effects really clean, and they're easy to use (like the rest of Vue).

Except when you try and use them on an array or object.

Nested data structures like arrays and objects are a little trickier to work with.

Just yesterday, I spent at least 30 minutes helping a co-worker figure out an issue with his Vue component.

Turned out to be an issue with nested data .

Because it’s such a common problem, I’ve written an in-depth article on how to watch nested data, which also goes into some of the more advanced features that watchers give you.

18. Show loading and error states on your async components

You probably hear a lot about web performance these days — and for good reason.

The easiest way to get your application bundle smaller is to split out the code into multiple, smaller, chunks. **Vue comes with first-class support for this**, which is really cool!

But the user’s experience can suffer if we don’t provide a good loading state while fetching the component. And we also want to show a good error state if something goes wrong.

Luckily, integrating these isn’t too difficult.

Joshua Bemenderfer shows us exactly how this can be done in Showing Loading & Error States with Vue.js Async Components.

19. Clean up your props for goodness sake!

Some components only require a few props, but others require passing many, many, props.

Eventually this can get pretty messy.

```
<v-btn
  color="primary"
  href="https://michaelnthiessen.com"
  small
  outline
  block
  ripple
>
  Hello
</v-btn>
```

But there are several different ways that we can clean this up. Not only will this make our code easier to look at, but it will also be easier to understand, and modify in the future.

Alex Jover Morales has written an excellent article outlining the different ways you can clean up your props. Check it out: Passing Multiple Properties to a Vue.js Component.

20. Don't confuse computed props and watchers

I know that most people don't accidentally write a computed prop when they meant to write a watcher.

That’s just silly.

But I see lots of people who use a watcher when they should instead be using a computed prop. Or using a computed prop when a watcher would be a better fit.

Although they seem like they do similar things, watchers and computed props are actually quite different.

My rule of thumb: make it a computed prop!

However, if you want to know more, I wrote an article on the differences between computed props and watchers.

21. Beware of some common pitfalls

Like any piece of technology, Vue has some areas that can catch you off guard.

I cannot tell you how many hours I wasted because I didn’t understand some of these things. But learning them didn’t take that long either.

If only I had known!

Instead of struggling through these gotchas like I did when I was learning Vue, you can avoid most of these.

Read Common Vue.js Gotchas by Joshua Bemenderfer, and save yourself a lot of frustration!

22. Learn the differences between props and data

Vue comes with two different ways of storing variables, props and data.

These can be confusing at first, since they seem like they do similar things, and it’s not clear when to use one vs the other.

The answer involves reactivity, naming collisions, and the direction of data flow (spoiler: it’s down).

In my article on the difference between props and data, I also go into detail on where you would use each one, and how you would use them together.

It’s a really important topic to grasp, so make sure you understand it!

23. Properly call methods when the page loads

It’s an extremely common pattern in web apps to perform some sort of logic as soon as the page is loaded. Often you’re fetching data, or even manipulating the DOM somehow.

But there are a lot of *wrong* ways of doing this with Vue.

Lucky for us, Vue gives us lifecycle hooks that let us do this in a really clean and simple way.

You can check out this in-depth article on how to do this the proper way. The article also goes deep into what lifecycle methods are, and how we can hook into them.

24. Understand how to pass a function as a prop

Short answer: **you don't**.

But that's a hugely unsatisfying answer, so of course I'll expand on it.

This question comes up for 2 main reasons:

1. You want to communicate from the child to the parent
2. You need to abstract your component behaviour in a specific way

In React we pass functions around all the time, and that's how we would solve both of these problems. But Vue gives us two separate mechanisms for solving these two problems.

These mechanisms are events and scoped slots.

If you want to learn how to use these to solve either of these problems, as well as the differences between how React and Vue work (and more!), check out this detailed article I wrote about it: How to Pass a Function as a Prop in Vue

25. Learn why mutating props is an anti-pattern

This is an error you may have seen:

> *Avoid mutating a prop directly since the value will be overwritten whenever the parent component re-renders. Instead, use a data or computed property based on the prop's value.*

Why is mutating a prop directly not allowed in Vue, and what do you do instead?

I've written an entire article on this subject. I also go into what causes this error, and how to avoid it.

The article also touches on how to avoid getting this error when using `v-model`, as there are some specific things that can be confusing there.

26. Dynamically add CSS classes

Vue incorporates HTML and Javascript together in a really beautiful way, but we can't forget about CSS.

CSS is the thing that really makes our apps shine, and is very powerful in it's own right.

A very common pattern in web apps is to add and remove classes from elements based on the state of our application. We do this to show a button is disabled, to animate elements like loading spinners, and a ton of other things.

Vue gives us a lot of options in choosing how to dynamically add and remove CSS classes based on what's going on in our application. **Knowing what these options are gives you more tools, and you'll be able to write better code because of it**.

I wrote an article that covers all of the different ways you can dynamically add and remove classes in Vue. We go over array syntax and object syntax, using Javascript expressions to calculate the class, and adding dynamic classes to custom components (you don't need to add a custom `class` prop!). You can even generate your class names on the fly!

If you enjoyed this article, please share it with others who may enjoy it as well!

It really helps encourage me to keep writing stuff like this.

Thanks!