Framework Agnostic Web Development

Rick van der Eijk

The world of frontend development has undoubtedly become more complex over the years, and a significant part of this is due to the rise of different frontend frameworks. Because projects usually have a deadline, learning new frameworks is rarely convenient, which can be a huge problem for devs like me who want to broaden their skills.

The phenomenon can lead to decreased company effectiveness as well, which means that it is worth addressing.

Framework agnostic programming forms a solution to the above, as it is a way of working that frees us from the constraints of any particular framework. It allows us to be effective no matter the stack we work with.

Framework agnostic programming is about following timeless programming principles and erecting a decent architecture over adding countless libraries and following the latest fads, amongst other things.

Here are ten basic principles we can use to carve out such an approach.

Use frameworks only where it makes sense

Frontend development consists of two equally important but very different parts. The first is UI/UX while the second is data management. A good framework can certainly help you out with the former, but it won't necessarily

aid you with the latter. For example, while a tool like Redux can help you manage your state, it also adds another layer of complexity, which goes against the framework agnostic way of thinking. Redux toolkit is a step in the right direction, but is it really the best we can do?

We don't have to use frameworks for everything. The creators of such frameworks might well attempt to convince us otherwise, but what else would we expect from them? People still use vanilla Javascript to build websites even today, and with considerable success. While I'm not necessarily saying that this is the best approach for all projects, it is certainly still an option.

Use your framework for the parts of your app that need it, not for your entire application just because they told you so. It is up to us as developers to make good final decisions in these matters.

Create small components that are reusable throughout your app

A major advantage that frameworks offer over vanilla Javascript is code reusability. If you have 5 forms in your application with 10 fields each, you don't really want to type out the code for each form field in full. What you want is to set up a component for each different type of form field and then reuse them wherever they are needed.

The same goes for buttons, headers, footers, navigation elements, and so forth.

This rule is especially easy to violate in Angular because that framework comes with a lot of boilerplate. Not only do you need to add your components via the CLI but also you have to add them to the correct modules and route modules. It's a hassle and probably not as efficient as it could be. Nevertheless, the official Angular docs still state that small reusable components are the way to go, just as the official React docs do.

I would even go a step further and ask you why you're using a frontend framework at all, if you're not reusing your smallest components effectively. You certainly wouldn't be taking advantage of one of your frameworks main selling points, so why bother at all? Regular Javascript would work equally well for you, I'm sure.

The power of frameworks lies in the fact that we can structure our code in components, which we can then reuse wherever we want.

Pass only a single prop to child components

An important rule in programming in general is that we want to decouple our components and classes as much as possible. Some communication will always be necessary, but it should be both structured and sparse. Doing so will isolate the different parts of our applications, thereby making them easier to manage and easier to test.

Doing so in frontend applications will make your html templates or JSX a breeze to follow as well.

Obviously, we might have a dozen or more properties to pass to our child components, but that doesn't mean we have to pollute our templates with them. Rather, we can pass a single object and abstract the implementation details away. If we use TypeScript, we can make a new type for each data model we have, thereby allowing our code to become even cleaner.

Passing only a single object with properties to children is a good programming principle that will always make our lives easier.

Don't use complex libraries for state management

This might come as a surprise to many veteran Angular devs, but you can easily construct an entire state management solution using nothing but

regular Angular services. Moreover, such an approach is easy to build, a dream to follow for other developers, and infinitely scalable. I know because I've done that for real clients. RXJS is the only additional tool that we need, but it's been integrated into the Angular framework so well that it really is a natural part of it.

This completeness makes Angular gel extremely well with the framework agnostic approach. Yes, we are still using Angular for our data management, but we are using it in such a way that the approach could be easily copied to the framework of our choice.

React is a bit trickier in this regard as hooks are far from a perfect solution. Hooks are relatively opinionated and can be quite complex to manage as they have a number of limitations. Moreover, they don't tend to scale well, which is why state libraries such as Redux have remained popular even in recent times.

React does have options when it comes to the framework agnostic approach, but it requires us to look at external libraries. These don't have to be complex or opinionated, thankfully, but they do add a little bit of complexity for us to manage.

In summary, choose your state management solution wisely, if you need one at all, and make sure that it is both simple and opinionated.

Use CSS Grid for styling

CSS used to be tricky to write, as it had a tendency to become convoluted. That problem has been solved for many years, though, as CSS Grid makes it quite easy for us to style apps. It does have a bit of a learning curve, but not nearly as much as you might think.

The beauty of CSS Grid is that it can be used in any project at any time, regardless of the framework that was chosen.

CSS libraries such as Bootstrap and Angular Material do often come with pre-made components, which can save us a bit of time, but they also come with a learning curve of their own, forcing us to dig into our framework of choice just a little more.

In my experience, the tradeoff usually isn't worth it.

Conclusion

In part two of this article, we will take a look at five more framework agnostic principles that we can use to stand above the intricate world of frameworks with a holistic approach.