Let's talk about JavaScript frameworks

Victor Ndaba
Full stack web and mobile developer

To build a website, you only need three things

Things are a bit more complicated

- ➔ HTML Doesn’t scale well. You end up with a lot of repeated markup.
- ➔ CSS is not the easiest thing to learn (Naming conventions).
- ➔ We need a way to share and use packages created by other developers.
- ➔ Things get even more complicated when you need to incorporate third party services/APIs into your website/webapp.
- ➔ It becomes apparent that to build highly dynamic and scalable web apps, you need a better solution

Concept 1: Module bundlers

When writing code during development, we exercise “Separation of concerns”. This means that you’ll probably have multiple javascript files in a single code base which are linked to each other.

However, the browser’s JS engine can only read from a single source. It doesn’t understand “import” statements natively. There needs to be a way to reconcile this difference and perhaps “bundle” multiple files into a single source.

Modern browsers can understand import statements nowadays but are still unable to resolve external deps which is another reason we need bundlers

Concept 2: Polyfills and transpilation

The JS language spec is updated every now and then to introduce new features. Different browsers implement these feats differently and older ones may not support them at all. Polyfills present a common API for the developer and not have them think about how it's all natively implemented.

You won't normally work with polyfills directly; this is the job of a transpiler like Babel. It "transpiles" your code to make sure it runs across all browsers without you having to worry about it. To see the implementations of JS feats in browsers, checkout https://caniuse.com.

ECMAScript is the JS standard. It is maintained by TC39 and hence the versions of JS names: ES5, ES6, ES2022… etc.

ECMAScript and history of JS

JS was written in 10 days by Brendan Eich and it was meant to help create more dynamic content for browsers like Netscape navigator. It started out as Mocha, then LiveScript and eventually Javascript.

However, in its early days, there was no rigorous JS spec or standard. Everyone(browsers) implemented JS differently via different engines. Things were getting out of hand until the task of standardizing JS was given to the ecma foundation.

And thus, ECMAScript was born. This is why different versions of JS are named ES3, ES5 etc. In 2008, V8 was born which made JS fast and things just took off from there.

The birth of Client Side Frameworks

After the creation of node.js in 2009 by Ryan Dahl, JS frameworks started becoming more popular. Early proponents were AngularJS(angular 1) and backbone JS.

It was now possible to write JS everywhere and use it on the browser as well as on the server. These frameworks only kept growing, each offering its own features and insights and competing with the rest.

But there's a difference between a framework and a library. A library is more flexible; it doesn't change the way you structure your codebase(e.g React). But a framework has certain rules that you have to follow and changes how you structure your code(e.g Angular)

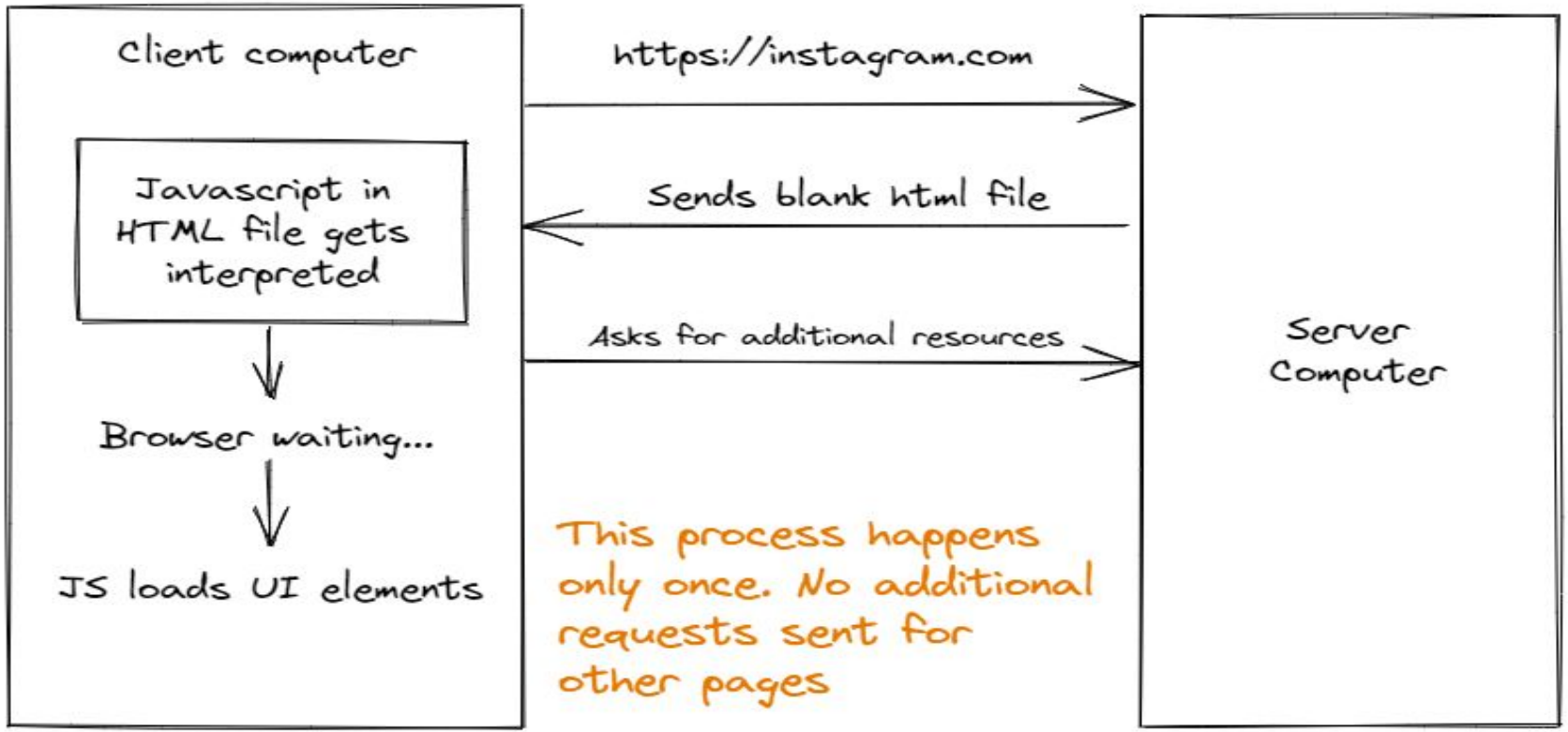
Concept 3: JavaScript frameworks

JS frameworks combine all these concepts that we've discussed. They often come with Command Line Interfaces(CLIs) that you use to bootstrap the applications and it comes with everything you need.

These frameworks abstract most of their build details but under the hood, they are using module bundlers and transpilers which make your overall developer experience very enjoyable.

These frameworks are called Client Side Frameworks and are used to build Single Page Applications. These are so named because, only one html file is requested from the server and from there, JS takes over creating the "illusion" of a multi-page app.

Client Side Rendering

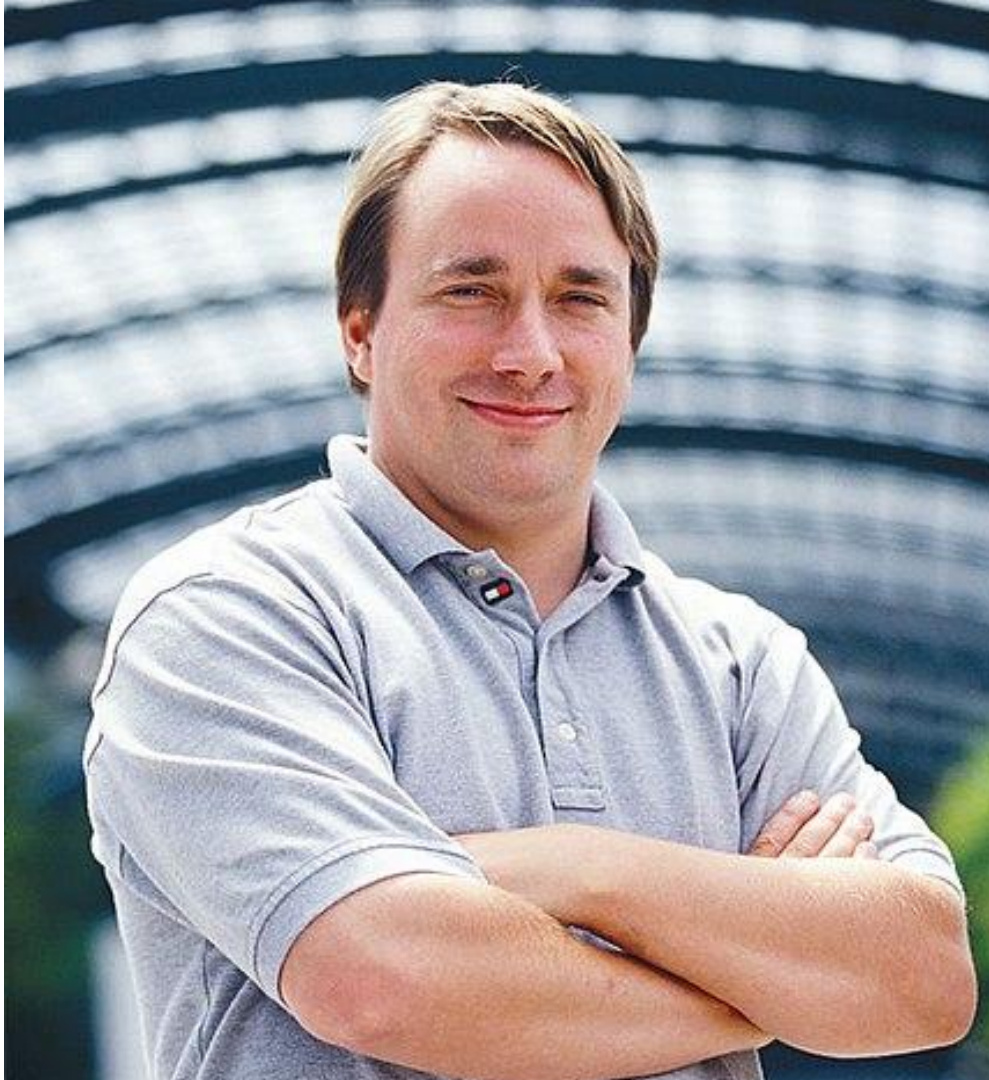


JS frameworks concepts

- Components - A self contained piece of code that is used to prevent code duplication
- State - This simply refers to the “memory” that is “held” by a component. It can be local(for a single component) or global(shared across multiple components)
- Component Lifecycle - The “lifecycle” of a component from when it is created to when it is destroyed
- Additional concepts like Routing, Building, Deployment etc.

Talk is cheap, show me the code.

~Linus Torvalds



General tips

- Use pnpm as your package manager
- Use tailwindcss for a framework and if you have to use a component library, you should try out mantine
- Use vite or modern bundlers over CLIs like create-react-app
- JS ecosystem is changing crazily, stay informed via youtube academy. Fireship, Theo, etc…
- ABC(Always be coding)