WebAssembly and the Wizards of Hogwarts

Daniel Burger

7. November 2020

Abstract

There is a new kind in the web development hood: WebAssembly. It is fast, portable, supported by the big players and should allow making the world wide web the biggest software platform in existence. It doesn't matter if you are an experienced software architect or a front-end developer — everyone will be able to profit from WebAssembly's existence.

In this article, we will look at what WebAssembly is, how it works and how it works alongside its sibling JavaScript.

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Figure 1.1: Harry Potter dress form with a black jacket (Unsplash, 2018)

On the 17th of June 2015, Brendan Eich — the inventor of JavaScript — and the teams behind Mozilla, Chromium, Edge and WebKit presented a new browser standard: WebAssembly, a portable and highly efficient byte-code compilation target for high-level languages such as C++ and Rust (Eich, 2015).

However, what does this mean? What is the reason that WebAssembly should exist in the first place? Should JavaScript developers be worried now? And what do the wizards of Hogwarts have to do with it?

1.1 Web Development Back Then

Back in the day when you could call yourself web developer because you only understood HTML, web development itself was a rather interactionless and static field of business. Netscape Communications, one of the establishing companies who built the world wide web as we know it today, soon realised that websites lacked to be interactive and dynamic (Cassel, 2018). They wanted the web to be a new form of a distributed operating system rather than just a simple HTML document-accessing application on your computer.

Marc Andreessen, who was the founder of Netscape Communications, proposed that HTML needed some kind of "scripting language" that was approachable — something you could glue directly into the markup. A language that is easy to use by newbie programmers who didn't want to handle compiler errors or strictly-typed syntax.

Figure 1.2: Screenshot of the Netscape start-page (NPR)

That was the reason they hired the experienced programming language and network code developer Brendan Eich. Brendan's first task was the nearly unachievable goal of creating such a scripting language for the web — due in 10 days. They (later) called it: JavaScript (Severance, 2012).

1.2 JavaScript's Destiny

I don't need to dig too deep into the history of the web to show you one crucial pain point of today's web technology standards: JavaScript is a scripting language for the browser to interpret. I repeat it: JavaScript is a scripting language for the browser to interpret — not some fancy multi-paradigm system programming language that focuses on speed, security or code maintainability. It was supposed and designed to be an easy-to-understand dynamically-typed scripting language to give your website some cool DOM manipulations and decorative animations.

Nevertheless, see what happened:

> *Any application that can be written in JavaScript, will eventually be written in JavaScript. – Jeff Atwood*

A quote by Jeff Atwood (co-founder of Stack Exchange) that is popularly referred to as Atwood's Law (Atwood, 2007). Ever since frameworks like NodeJS or React Native became widely used, JavaScript's possibilities already crossed the border of just living on the client-side inside of a browser. It's nearly everywhere.

Also, NodeJS' NPM package manager is currently the most prominent and most active package registry platform ever created. As an example: From May 10th to May 17th of 2018, JavaScript developers downloaded 5.2 billion NodeJS packages from the NPM registry, setting a new record (Inc, 2018)

Figure 1.3: Google Trends curve about NodeJS and React Native

1.3 Treacherous Atwood's Law

There are thousands of courses, books and tutorials on how to learn JavaScript. Every computer-related school now or then teaches JavaScript. Nearly everyone could be able to learn it anywhere with a minimum amount of effort. If you search for "Learn JavaScript" on Google, you'll get 2 220 000 000 search results. In comparison: When you search "Learn Java" you'll get 305 000 000 results.

Though, is that a good thing? Is an originally lightweight scripting language capable of ruling the world of computational web development? My opinion: No, it's not, and I believe there won't be such a bright future for JavaScript as nearly everybody claims. Let me explain it with a Harry Potter analogy:

1.4 Muggles Entering Hogwarts

Do you know how it feels to watch front-end developers call themselves "full-stack software developer" after they've simply learned NodeJS? It feels like Muggles entering Hogwarts—a school full of wizards. Only that in our case these wizards are trained software engineers and computer scientists. These are the people who learn all the hardcore-implemented algorithms and compilers, programming languages and operating systems. They know precisely how to build software (Might, 2011).

> *Do front-end developers know how to write actual software? I'd say we better don't talk about it.*

Muggles aren't invited to Hogwarts because they have no magical ability. Front-end developers weren't "invited" for software engineering too because they couldn't even do something with the languages they knew. Nevertheless, now — thanks to all the cross-platform JavaScript runtimes — they're suddenly here to do some magic. But imagine, what if the wizards of Hogwarts, our beloved software engineers and computer scientists, would be able to perform magic in the real world — or our case: the front-end? What would happen? What could go wrong?

Figure 2.1: Illustration of how WebAssembly modules are being delivered to the browser (LogRocket)

2 WebAssembly Says Hello World

WebAssembly is the new player in the web development industry. It's fast, small, non-readable and not even a real programming language. Yes, you heard that right. You literally can't code in WebAssembly (Rourke, 2018). So I may hear you asking: Why should we all be excited about it? Well, as mentioned earlier, it's a compilation byte-format target for high-level languages. You write your code in such a high-level language like C or C++ and compile it down to WebAssembly. The magic trick: It works in the browser, and it's super fast — sometimes about 5 to 20 times faster than JavaScript (Aboukhalil, 2019).

2.1 WebAssembly in a Nutshell

Enough with the marketing fuzz. The real face behind the term "WebAssembly" isn't that uniform as it's being marketed. WebAssembly itself is only a piece of a bigger technology chain of workflows and concepts. There are several other key components that are important for delivering super-fast web applications. It's also good to know what its current limitations are and for which use cases it's the best. But first, let me introduce you to the five key components:

2.2 WebAssembly in a Nutshell

This is a human-readable file format you'll get when you compile your C, C++ or Rust code. It represents the abstract syntax tree (AST) from the source code of a programming language. An AST — or in the WebAssembly case: a .wat file — may be verbose, but it does an excellent job at describing the components of source code. Representing source code in an AST makes verification and compilation simple and efficient. Here is a simple return function called `getDoubleNumber` written in C:

```
int getDoubleNumber(int number) {
  return number * 2;
}
```

Listing 2.1: Python example

Bibliography

Aboukhalil, R. (2019). How We Used WebAssembly To Speed Up Our Web App By 20X (Case Study). Publication Title: Smashing Magazine.
URL https://www.smashingmagazine.com/2019/04/webassembly-speed-web-app (Accessed at: 2022-12-11)

Atwood, J. (2007). The Principle of Least Power. Publication Title: Coding Horror.
URL https://blog.codinghorror.com/the-principle-of-least-power (Accessed at: 2022-12-11)

Cassel, D. (2018). Brendan Eich on Creating JavaScript in 10 Days, and What He'd Do Differently Today. Publication Title: The New Stack.
URL https://thenewstack.io/brendan-eich-on-creating-javascript-in-10-days-and-what-hed-do-differently-today (Accessed at: 2022-12-11)

Eich, B. (2015). From ASM.JS to WebAssembly – Brendan Eich.
URL https://brendaneich.com/2015/06/from-asm-js-to-webassembly (Accessed at: 2022-12-11)

Inc, n. (2018). How well do you know your npm trivia? Publication Title: Jaxcenter.
URL https://jaxenter.com/npm-trivia-144846 (Accessed at: 2022-12-11)

LogRocket (n.d.). LogRocket Blog. Publication Title: LogRocket Blog.
URL https://blog.logrocket.com/ (Accessed at: 2022-12-11)

Might, M. (2011). What every computer science major should know. Publication Title: Matt Might Blog.
URL https://matt.might.net/articles/what-cs-majors-should-know (Accessed at: 2022-12-11)

NPR (n.d.). Home Page Top Stories. Publication Title: NPR.org.
URL https://www.npr.org (Accessed at: 2022-11-19)

Rourke, M. (2018). *Learn WebAssembly: Build web applications with native performance using Wasm and C/C++*. Packt Publishing.

Severance, C. (2012). JavaScript: Designing a Language in 10 Days. *Computer*, *45*(02), 7–8.
URL https://www.computer.org/csdl/magazine/co/2012/02/mco2012020007/13rRUy08MzA (Accessed at: 2022-12-11)

Unsplash (2018). Beautiful Free Images & Pictures \textbar Unsplash. Publication Title: Unsplash.
URL https://unsplash.com (Accessed at: 2022-11-19)