WebAssembly: Constructing Versatile Programs with Multiple Programming Languages

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia - Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Alan Li

Spring, 2023

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISORS

Joshua Earle, Department of Engineering and Society

Briana Morrison, Department of Computer Science

ABSTRACT
An Arlington, VA-based cloud computing company required a web coding playground for one of its new services enabling users to input code and evaluate its validity and output. However, some customers required a local application for their confidential data. I was tasked with designing and building a React.js web app to serve as the frontend user experience but required a backend code evaluator without sending data to and from the customer and developers. The solution was to use WebAssembly to compile backend Rust code to an assembly and JavaScript package which was then used in the React.js web app to evaluate code locally.

With these two components connected, the app was fully functioning locally, without a need for any network calls. The result was a much faster application as initial evaluation calls were 50 times faster and subsequent calls 100 times faster than a similar playground using API calls. While the application was complete, many user interface designs were not fully implemented, and an online hosted version required for internal use had not yet been completed.

1. INTRODUCTION

Traditionally, complex application logic is evaluated through more robust languages such as Java, C++, and Rust which then exports its results to web applications built mainly in JavaScript. This is typically built with network calls (i.e. API or Application Programming Interface calls) to a developer-managed server that compiles code and then returns data to the web app.

According to Akamai Staff (2019), API calls have made up 83% of the web since 2018. Amoroso (2020) points out that this percentage continues to grow as cloud-based applications continue growing, and with this expansion comes a number of security issues. These include injection attacks and authentication errors as users can send data to the servers that handle the API calls or bypass the authentication tokens needed to access the API network.

If API calls are so bad, then why not ship applications with all the backend logic baked in? The two issues with this are that browsers are not built to support heavy-duty languages such as Java or Rust and even if possible, the web app would become much too large to run in a browser. The security/confidentiality and efficiency benefits that come with using WebAssembly have made it a very enticing tool for many developers who use it for heavy-duty operations on the web previously too inefficient such as heavy asset games and CAD.

2. RELATED WORKS

My first encounter with WebAssembly was during my project when I considered it as a potential solution. I have since learned much more about the potential problems that it has been used to solve. Carey (2022) talks about the history of WebAssembly and a few examples of companies with large customer bases using it for major products. Inspiration came from the innovative technologies that are being built with WebAssembly.

WebAssembly has had many other uses as McCallum (2019) discusses in some detail. He covers the growing list of applications it is being used for, providing many use cases and exploring how WebAssembly can be used to benefit current and future web development projects.

3. PROJECT DESIGN

As this was a new project being created for numerous users, requirements elicitation

was crucial to ensure a good overall design. Thus, I spent time talking to multiple teams and product managers to ensure my requirements met all customer needs and that the design I created met company standards.

3.1 Requirements

The main purpose of this project was to create an easy-to-use web app that customers can utilize to process their data using the already-built Rust code developed by the development team. Along with this is the need for customer data to be confidential as there were potential customers that would not want to send their data to developers.

3.2 Design Overview

The two main components to this project were a frontend web application and the logic used to evaluate data which is written in Rust. I built the frontend web app in React.js, currently one of the most popular web development frameworks (Stack Overflow, 2022), as it was considered a standard within the company and developers were already familiar with the framework. This enabled easier development as the team was more familiar with the process and would be able to maintain and debug issues that came up in the future. Unlike the web app, the data processing code was fully developed and ready to use. As this code was written in Rust, it could not be directly run through the browser and thus required a different solution to ensure the code was usable in the web app.

3.3 Complications

Designing the web app was a straightforward process but integrating the data processing feature required a new solution as Rust and most other high-level programming languages are not natively supported on most web browsers. This led to discussions of setting up a cloud server to handle data processing from the web app as customers would be able to input their data in the app which would send the data to the developer-maintained cloud server.

However, this brought about two issues. The first is cloud servers rest in a dormant state to save processing costs and initial calls to it would require the server to boot up with each subsequent call taking a noticeable amount of time to process. This process has been used many times in the past so it has been very well documented but can be a bit slow for an app that should be fast and simple to use. On top of that, this approach would require customers to always send their data to the developers which would mean a portion of the customers with the confidential information requirement would not be able to use this app.

The second option would be to bake the Rust code inside the app but this leads to the second issue as most browsers do not support Rust. An installable application could be created for customers to download but this would go against the “easy-to-use” requirement as utilizing Rust would require customers to install the language itself and all the packages used which can be overwhelming for someone who is not very familiar with the language.

3.4 Solution

These complications led to my intern project of researching how WebAssembly could be used to meet both of these requirements. In short, WebAssembly compiles code such as Rust into low-level assembly-like code – WebAssembly modules – which can then be run alongside JavaScript which is used for most of the internet’s functionalities (Webassembly MDN., 2022). This process requires installing the WebAssembly package called wasm-pack and then annotating lines of Rust code that will be

called directly in the web app through its JavaScript code. While the rest of the Rust code will be compiled into the WebAssembly module, the lines of code that are annotated will be converted into JavaScript which can then be called directly from the web app.

Finally, once the Rust code had been compiled, I imported the WebAssembly module into the React.js web app. I accomplished this by directly moving the created WebAssembly module into the web app's directory and calling the converted JavaScript code to utilize the Rust code's functionality.

4. Results

The resulting web app was a lightweight app that required no external server to utilize its full functionality. Compared to a similar data processing app, the new app was able to process data approximately 50 times faster on initial calls and 100 times faster on subsequent calls. For customers, this meant an app they can use just by navigating to a website or downloading a small package of JavaScript files and WebAssembly modules to run and utilize on their own computers without navigating to the website. Utilizing WebAssembly enabled the project to meet all customer needs without sacrificing speed or project quality.

5. Conclusion

WebAssembly has been gaining popularity since its inception and this project shows how versatile it can be in building complex software applications. Exploring the applications of WebAssembly for my project resulted in a much faster application and enabled many more customers to use it. But these benefits are only the beginning of the potential WebAssembly can bring to web development and as more developers begin using it, its versatility as a tool enabling developers to find new ways to solve problems will only grow.

6. Future Work

While I was able to create a working web app that could be sent out to customers, it was not yet hosted for users to navigate to on a browser. Thus, hosting the web app will be the next major step in creating the finalized product as a convenient-to-use web app, but users who want to run everything locally also need an option to download. Another issue that required some more thought was how to automatically compile WebAssembly modules every time the Rust code base was updated by developers as the current module is manually compiled and then imported directly into the web app. Outside of my project, I hope my project encourages more developers to explore how they can use WebAssembly to improve their current projects or help them create exciting and innovative projects in the future.

References

Carey, S. (2022, February 28). *The rise of WebAssembly*. InfoWorld. Retrieved September 30, 2022, from https://www.infoworld.com/article/3651503/the-rise-of-webassembly.html

McCallum, T. (2019, November 28). *The Future of Web Assembly (WASM): The hardware-execution revolution!* Medium. Retrieved September 30, 2022, from https://medium.com/wasm/the-future-of-web-assembly-wasm-the-hardware-execution-revolution-4116eafa39a0

Stack Overflow. (July 1, 2022). *Most used web frameworks among developers worldwide, as of 2022* [Graph]. In Statista. Retrieved November 01, 2022, from https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Webassembly. MDN. (2022, October 12). Retrieved October 27, 2022, from https://developer.mozilla.org/en-US/docs/WebAssembly