

Yew un framework WebAssembly & Rust en el Lado del Frontend

Frank Wilson Mamani Mamani¹ (0000-0002-4964-5294), Guido Yimi Curo Ramos² , Jonatan Huarahuara Pongo³ , Gian Grober Mamani Mamani ⁴ (0000-0003-1917-4070)

¹ Escuela Profesional de Ingeniería de Sistemas, Universidad Peruana Unión, Filial

Juliaca, Carretera salida a Arequipa Km 6 Chullunquiani, Juliaca, Perú

frank.mamani@upeu.edu.pe, guido.curo@upeu.edu.pe, jonatan.hp@upeu.edu.pe

Keywords(WebAssembly, Rust, WASM, Yew, Frontend)

1. Introducción

Desarrollo de la introducción

WebAssembly, o también conocido como WASM, es un formato de código binario portable de bajo nivel estandarizado alternativo a javascript que puede ser ejecutado en la mayoría de navegadores modernos. WASM está diseñado para ser un objeto final de compilación para lenguajes de bajo nivel como C, C++, C#, Rust, Go, etc. [1] Añadiendo nuevas funcionalidades y mejoras con respecto a velocidad, rendimiento y seguridad[5]. gracias a la escritura estática, la seguridad de la memoria, manejo de hilos entre otras ventajas heredadas de los lenguajes de bajo nivel.

Con la llegada de esta tecnología se abrieron un sinnúmero de oportunidades para que muchos desarrolladores en el mundo puedan utilizar casi cualquiera de sus lenguajes de programación favoritos en el navegador, fruto de ello nacieron librerías y frameworks como Blazor, Vuetify o Yew por mencionar algunas que son frameworks para el frontend muy modernos que usan lenguajes variados en el lado web del cliente gracias a WASM llegando a ser alternativas directas y quizás en un futuro posibles sustitutos de librerías-frameworks que solo usan javascript como angular, react, vue, etc.

Es por esta razón que basados en los resultados de algunas comparaciones realizadas entre JS y WASM además de la demostración realizada de la perfecta funcionalidad del framework Yew haciendo uso de Rust en el navegador, se darán algunas recomendaciones que permitirán a los desarrolladores motivarse y ver nuevas alternativas como futuras mejoras en cuanto a tecnologías web.

1.1 WebAssembly

Anunciado en el año 2015 [2] y en el 2017 fue implementado por todos los principales navegadores. [3] WebAssembly es compatible con el 92% de todas las instalaciones de navegadores globales a partir de junio de 2020. WebAssembly [8] es un

nuevo formato de código binario portátil, que además de mantener el modelo seguro y aislado que proporciona la web, aporta velocidades casi nativas a la web y sirve como un destino de compilación más apropiado para lenguajes mecanografiados como C y C ++. Por lo tanto, abre las puertas a una variedad de lenguajes diferentes y cierra la brecha en el rendimiento al permitir aplicaciones que antes eran difíciles de migrar a la web. Actualmente, el kit de herramientas Emscripten [9] proporciona un marco para compilar C y C ++ en WebAssembly junto con un entorno de ejecución integrado para WebAssembly en JavaScript que expone las funciones C y C ++ al tiempo de ejecución de JavaScript. WebAssembly se construyó como una abstracción sobre las principales arquitecturas de hardware, proporcionando un formato que es independiente del lenguaje, el hardware y la plataforma [13]. La naturaleza de bajo nivel del lenguaje debería ofrecer muchas oportunidades para optimizaciones que benefician los cálculos numéricos, al momento de escribir, la característica SIMD de ancho fijo y el paralelismo a través de subprocesos están en la etapa de progreso para WebAssembly [14]. Además, WebAssembly admite diferentes tipos de enteros y puntos flotantes de precisión simple, que actualmente no son compatibles con JavaScript.

1.2 Rust

Rust es un lenguaje de programación de bajo nivel desarrollado por los ingenieros de Mozilla. Podríamos decir que es un lenguaje aún muy nuevo; Estamos hablando que la versión 1.0 estuvo disponible a principios del 2015; Actualmente en la versión 1.65.0.

Al igual que C , Java o C++, Rust es un lenguaje compilado. Rust está diseñado para desarrollar software de sistemas, donde la interacción con el usuario es prácticamente nula, se han venido haciendo esfuerzos para llevarlo a otros entornos como lo es la web gracias a yew framework.

El éxito de Rust es sorprendente. Año tras otro las encuestas lo califican como el lenguaje que todo programadores ama, y con razón: tiene muchas de las ventajas de los

legendarios C y C++, pero sin sus problemas de gestión de memoria, podría quizás convertirlo en sucesor de esos legendarios lenguajes.

1.3 Yew

Yew es un marco Rust moderno análogo de React y Elm para crear aplicaciones web front-end de subprocesos múltiples utilizando WebAssembly

Cuenta con una macro para declarar HTML interactivo con expresiones Rust. Los desarrolladores que tienen experiencia en el uso de JSX en React deberían sentirse como en casa al usar Yew.

Logra un alto rendimiento al minimizar las llamadas a la API DOM para cada procesamiento de página y al facilitar la descarga del procesamiento a los trabajadores web en segundo plano.

Admite la interoperabilidad de JavaScript, lo que permite a los desarrolladores aprovechar los paquetes de NPM e integrarse con las aplicaciones de JavaScript existentes.

Es importante tener en cuenta que el ecosistema Wasm y Yew aún se encuentran en etapas de su desarrollo y no se recomienda para producción.

[11]

2. Metodología

El desarrollo del método está compuesto por 5 etapas generales como elección y conversión de algoritmos, despliegue de algoritmos WASM, las pruebas de rendimiento de los algoritmos, creación de proyecto yew y validación de correcto funcionamiento en el navegador web de este framework.



2.1 Elección y conversión de algoritmos

- a. Se efectuará la clasificación de algunos algoritmos que procederemos a convertir a lenguajes JS, C y WASM.

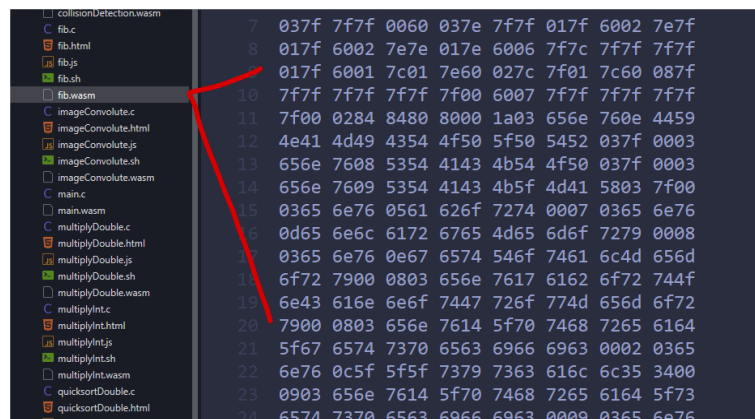
```
int fib(int n) {  
    if (n == 1) return 1;  
    if (n == 2) return 1;  
    return fib(n-1) + fib(n-2);  
}
```

```

1 #include <math.h>
2
3 struct position {
4     double x;
5     double y;
6     double z;
7 };
8
9 int collisionDetection(struct position *positions,
10                      double *radiuses,
11                      unsigned char *res, int n) {
12     int count = 0;
13     for (int i = 0; i < n; i++) {
14         struct position p = positions[i];
15         double r = radiuses[i];
16         unsigned char collision = 0;
17         for (int j = i+1; j < n; j++) {
18             struct position p2 = positions[j];
19             double r2 = radiuses[j];
20             double dx = p.x - p2.x;
21             double dy = p.y - p2.y;
22             double dz = p.z - p2.z;
23             double d = sqrt(dx*dx + dy*dy + dz*dz);
24             if (r > d) {
25                 collision = 1;
26                 count++;
27                 break;
28             }
29         }
30         int index = (i / 8) | 0;
31         unsigned char pos = 7 - (i % 8);
32         if (collision == 0) {
33             res[index] &= ~(1 << pos);
34         } else {
35             res[index] |= (1 << pos);
36         }
37     }
38     return count;
39 }
40

```

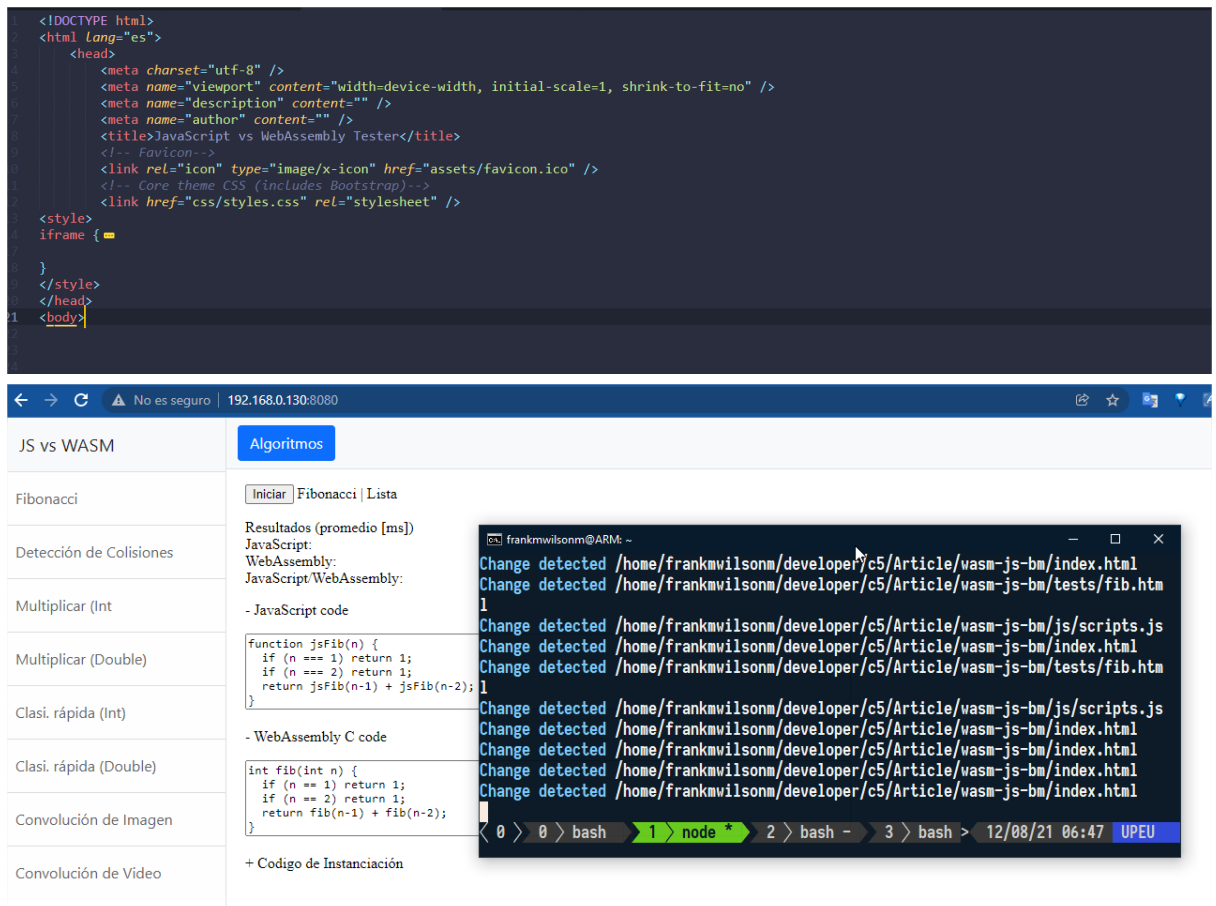
- b. Seguidamente con la ayuda de Emscripten que es un compilador para webassembly podemos convertir nuestros algoritmos de C a WASM, que contendrá los binarios que luego usaremos para instanciarlo en nuestro html.



- c. Se pueden encontrar muchos binarios WASM ya generados con Emscripten en internet, los cuales también podemos usar ya que instalar el compilador mencionado puede tomar algo de tiempo.[6]

2.2 Despliegue de algoritmos JS y WASM

- a. Se efectuará la implementación de nuestros algoritmos haciendo uso de frameworks web las cuales nos ahorran mucho tiempo diseñando interfaces web tales como bootstrap que es la que usaremos para este caso de uso.



The screenshot shows a web browser displaying a page titled "Algoritmos". The page has a table with the following content:

Algoritmo	Acción
Fibonacci	Iniciar Fibonacci Lista
Detección de Colisiones	Resultados (promedio [ms]) JavaScript: WebAssembly: JavaScript/WebAssembly:
Multiplicar (Int)	- JavaScript code
Multiplicar (Double)	function jsFib(n) { if (n === 1) return 1; if (n === 2) return 1; return jsFib(n-1) + jsFib(n-2); }
Clasi. rápida (Int)	- WebAssembly C code
Clasi. rápida (Double)	int fib(int n) { if (n == 1) return 1; if (n == 2) return 1; return fib(n-1) + fib(n-2); }
Convolución de Imagen	
Convolución de Video	+ Código de Instanciación

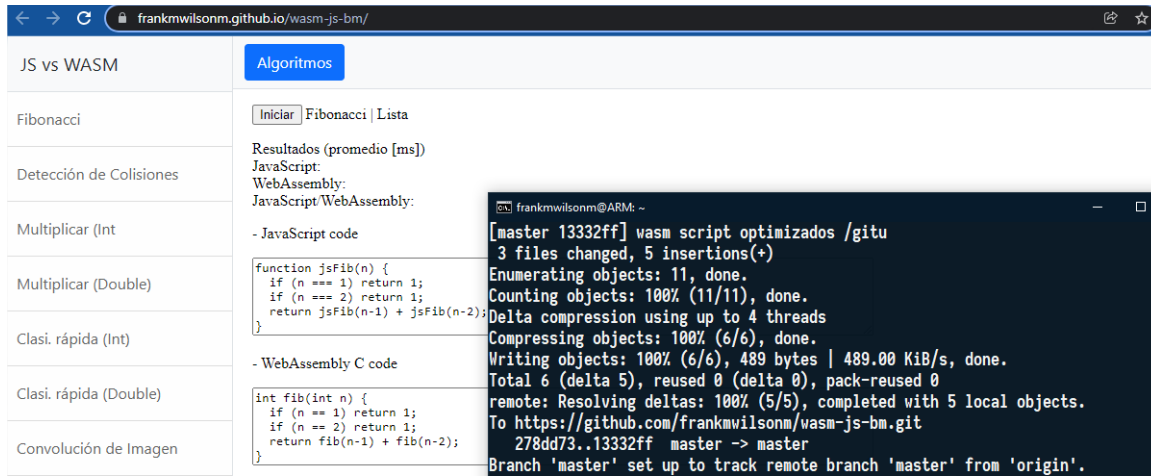
Overlaid on the right is a terminal window showing file change notifications:

```
frankwilsonm@ARM: ~  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/tests/fib.htm  
1  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/js/scripts.js  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/tests/fib.htm  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/js/scripts.js  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
Change detected /home/frankwilsonm/developer/c5/Article/wasm-js-bm/index.html  
0 > 0 > bash 1 > node * 2 > bash - 3 > bash > 12/08/21 06:47 UPEU
```

- b. Seguidamente vamos a instanciar nuestros algoritmos dentro de nuestro html.

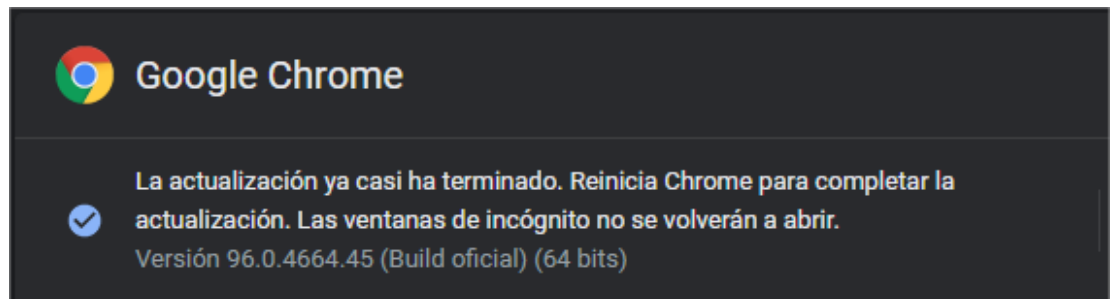
```
18  
19 <script id="ws_instantiate_code">  
20 var module, functions = {};  
21 fetch('fib.wasm')  
22 .then(response => response.arrayBuffer())  
23 .then(buffer => new Uint8Array(buffer))  
24 .then(binary => {  
25   var moduleArgs = {  
26     wasmBinary: binary,  
27     onRuntimeInitialized: function () {  
28       functions.fib =  
29         module.cwrap('fib',  
30           'number',  
31           ['number'],  
32             ['number']);  
33       onReady();  
34     }  
35   };  
36   module = Module(moduleArgs);  
37 </script>
```

- c. Wasm se puede implementar de muchas maneras y en seguida efectuaremos el despliegue web de nuestros algoritmos convertidos a javascript y wasm en github pages.

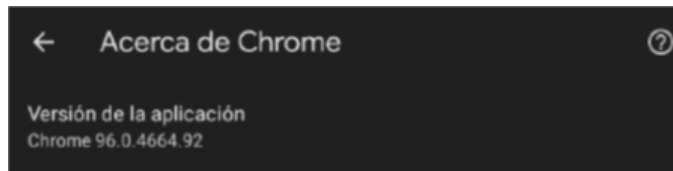


2.3 Pruebas de rendimiento JS - Webassembly

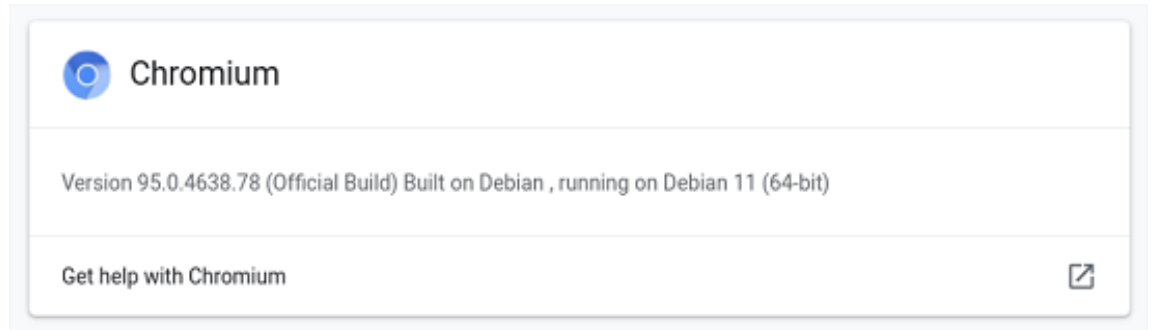
- a. En esta etapa vamos a iniciar con las pruebas de rendimiento con nuestros algoritmos ya desplegados en el servidor de github pages. [7]
- b. Para lo cual vamos a elegir el navegador chrome browser que es el que ejecutará los algoritmos para poder evaluar los tiempos de respuesta de javascript y wasm en milisegundos.
- c. En seguida vamos a elegir los dispositivos en los cuales vamos a analizar el rendimiento y la versión de navegador google chromium .
- Usaremos un computador que tenga las siguientes especificaciones: 4 x Intel Core i5 2.50GHz, 8GB DDR3, Windows 10, Google Chrome (64 bits) Versión 96.0.4664.45.



- También usaremos un dispositivo Android con las siguientes características: 4 x ARM Cortex-A53 MediaTek™ Helio A22 2.0GHz, 2GB LPDDR3, Android 10, Chrome 96.0.4664.92 Android Version.



- En representación de de las lot usaremos una raspberry pi 4 con las siguientes características: 4 x Broadcom BCM2711 A72 (ARMv8) 64-bit SoC @ 1.5GHz, SDRAM LPDDR4-2400 2GB, Raspberry Pi OS Lite, Chromium (64 bits) Version 95.0.4638.78.



2.4 Implementación de proyecto con Yew

- Antes de poder crear un nuevo proyecto en Yew necesitamos contar la instalación y con los requisitos mencionados en la documentación oficial de yew, la instalación toma algo de tiempo si es la primera vez que instalamos Rust <https://yew.rs/docs/getting-started/introduction>

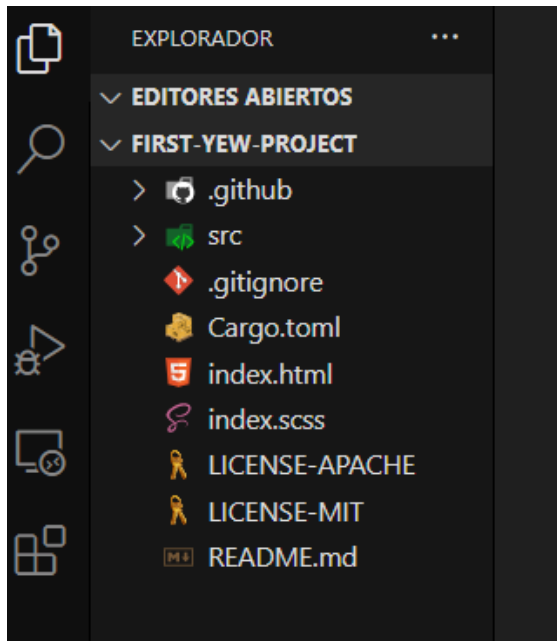
- Crearemos una aplicación de muestra que nos proporciona Yew:

cargo generate --git https://github.com/yewstack/yew-trunk-minimal-template

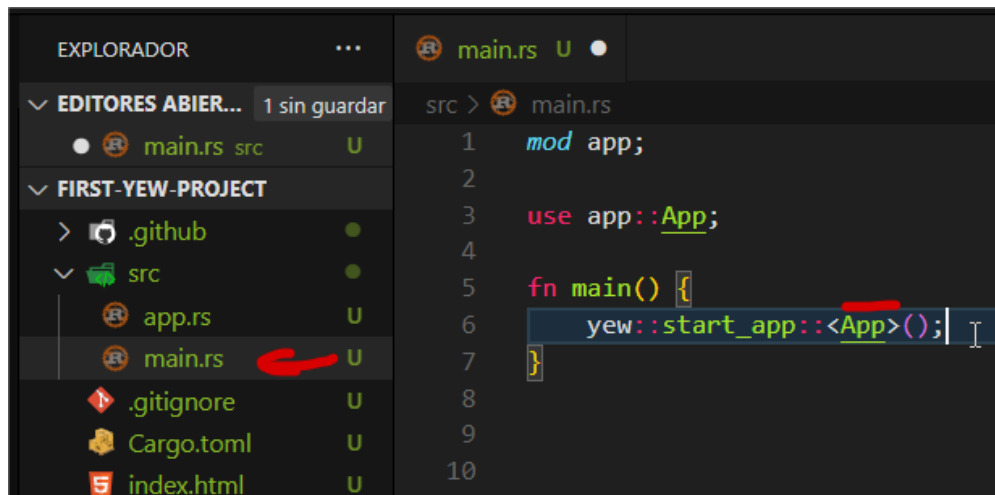
```
E:\developer\depruebas\yew>cargo generate --git https://github.com/yewstack/yew-trunk-minimal-template
Project Name: first-yew-project
```

- Abrimos nuestra carpeta de proyecto con cualquier editor de texto o código.

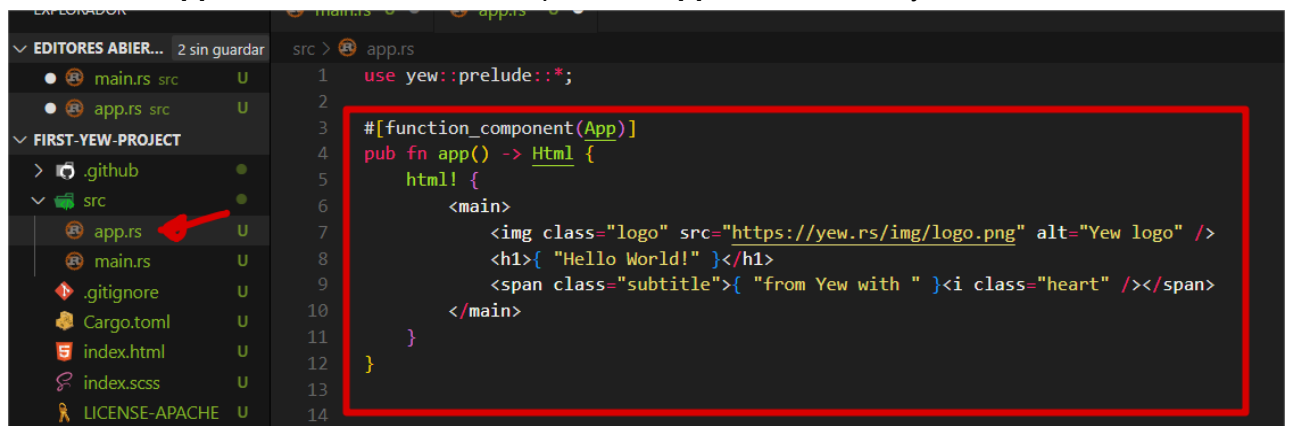
Obtendremos una estructura muy familiar a React js



También tenemos una función main que corre nuestro componente `<App>`



Y dentro de `app.rs` tenemos nuestro componente `App` tal cual `React` y `Jsx`

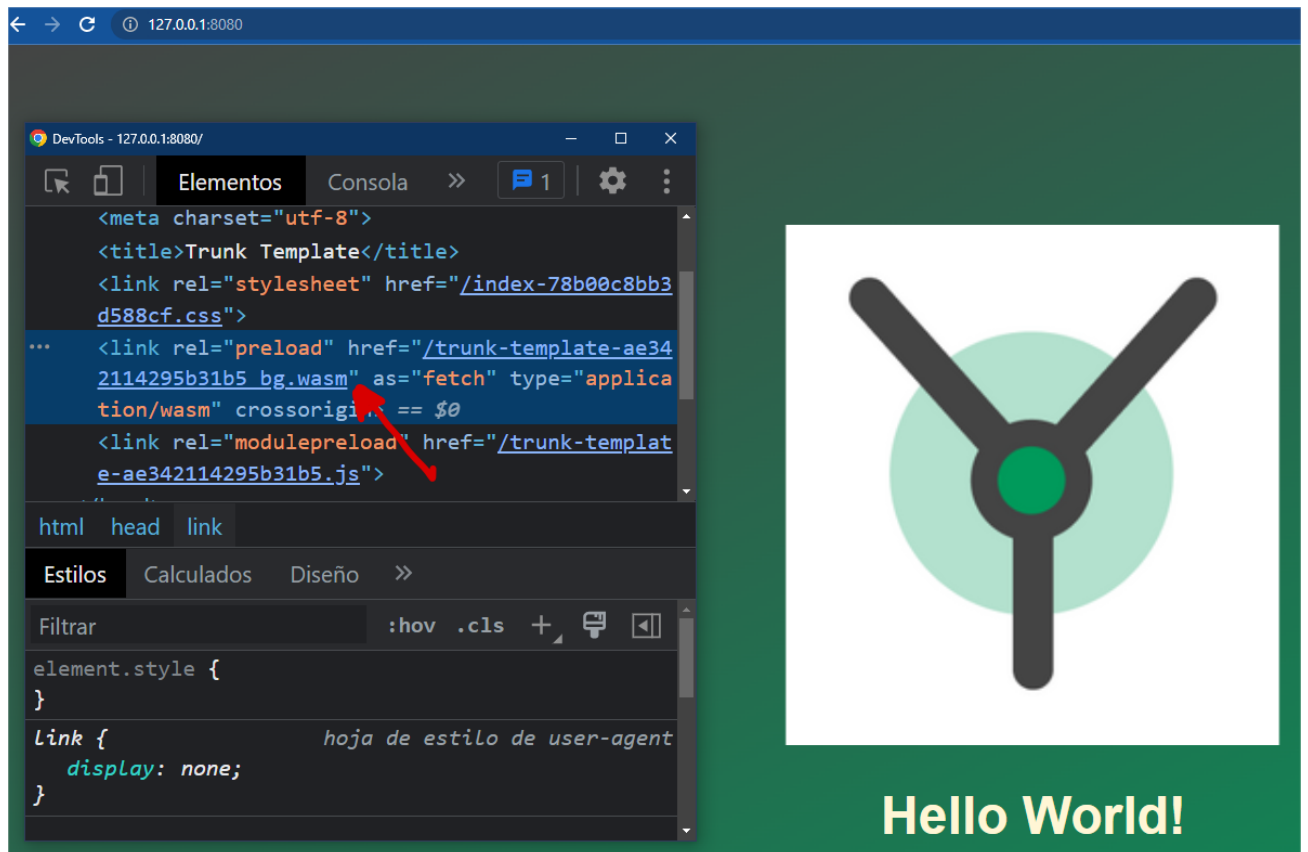


2.5 Demostración de funcionalidad del framework Yew en el navegador

- Iniciamos con **trunk serve** para encender el servidor de **Yew**

```
E:\developer\depruebas\yew\first-yew-project>trunk serve
2022-12-01T08:36:13.032868Z INFO starting build
2022-12-01T08:36:13.168390Z INFO spawning asset pipelines
```

Como verán nuestro proyecto de prueba a generado el front similar a React Js además de hacer uso del archivo binario **.wasm**



The image shows a browser window displaying a "Hello World!" message in a green font on a dark green background. To the left, the browser's developer tools are open, showing the HTML structure of the page. A red arrow points to a `<link rel="preload" href="/trunk-template-ae342114295b31b5_bg.wasm" as="fetch" type="application/wasm" crossorigin="" == $0` tag in the `head` section. The `as="fetch"` attribute is highlighted in blue. Below the HTML, the `Link` component's style is shown as `display: none;`.

Probaremos una vez más la funcionalidad pero ahora agregando un nuevo componente.

```
EDITORES ABIER... 2 sin guardar src > app.rs
1 use yew::prelude::*;
2
3 #[function_component(App)]
4 pub fn app() -> Html {
5     html! {
6         <main>
7             <h1>{ "Componente 1" }</h1>
8             <p>{ "Hello World!" }</p>
9             <Ncomp></Ncomp>
10        </main>
11    }
12 }
13
14
15 #[function_component(Ncomp)]
16 pub fn ncomp() -> Html {
17     html! {
18         <main>
19             <h1>{ "Componente 2" }</h1>
20             <p>{ "Hello Upeu!" }</p>
21        </main>
22    }
23 }
24
```

Visual Studio Code

```
src > app.rs
2
3 #[function_component(App)]
4 pub fn app() -> Html {
5     html! {
6         <main>
7             <h1>{ "Componente 1" }</h1>
8             <p>{ "Hello World!" }</p>
9             <Ncomp></Ncomp>
10        </main>
11    }
12 }
13
14
15 #[function_component(Ncomp)]
16 pub fn ncomp() -> Html {
17     html! {
18         <main>
19             <h1>{ "Componente 2" }</h1>
20             <p>{ "Hello Upeu!" }</p>
21        </main>
22    }
23 }
24
```

Componente 1
Hello World!

Componente 2
Hello Upeu!

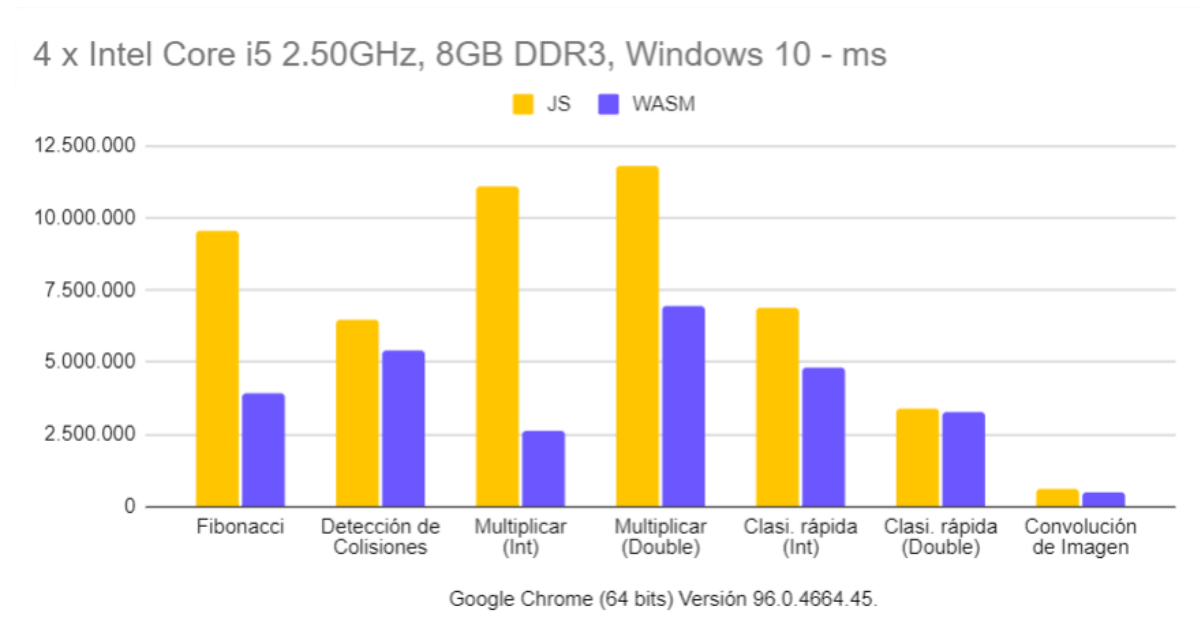
DevTools - 127.0.0.1:8080/

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Trunk Template</title>
    <link rel="stylesheet" href="/index-78b0c8bb3d588cf.css">
    <link rel="preload" href="/trunk-template-1bbe1075d8e06e3_bg.wasm" as="fetch" type="application/wasm" crossorigin == $0
    <link rel="modulepreload" href="/trunk-template-1bbe1075d8e06e3.js">
  </head>
  <body>
    <main>
      <h1>Componente 1</h1>
      <p>Hello World!</p>
      <main>
        <h1>Componente 2</h1>
        <p>Hello Upeu!</p>
      </main>
    </main>
  </body>
</html>
```

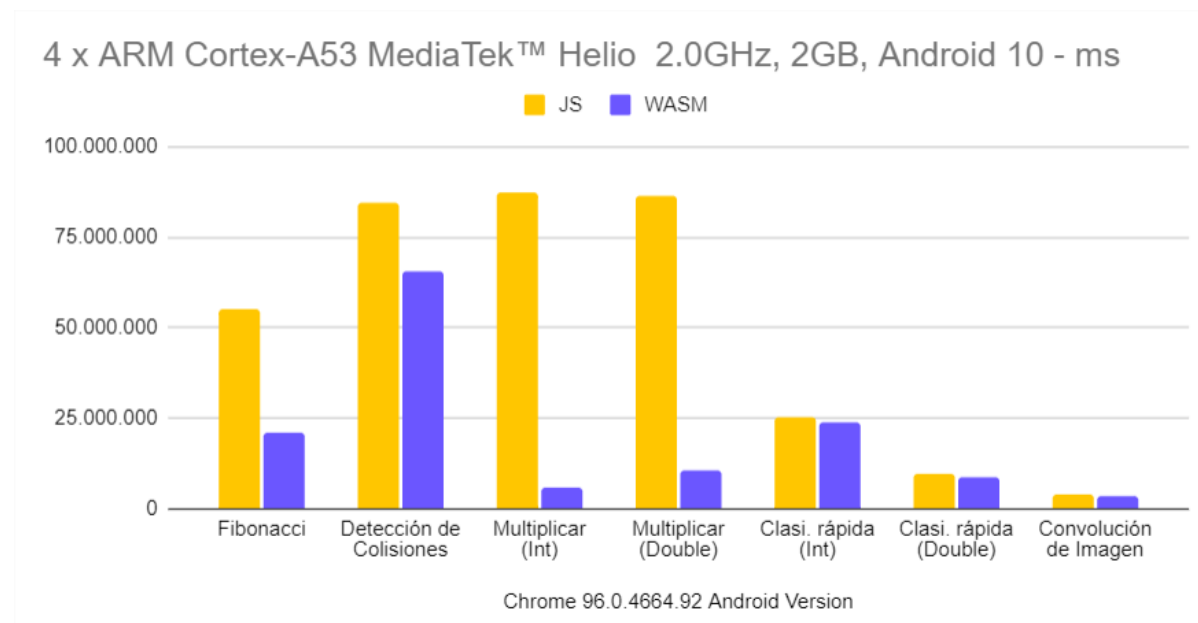
3. Resultados

Se obtuvieron los datos del análisis comparativo del rendimiento entre javascript y WebAssembly con las condiciones ya especificadas anteriormente teniendo resultados aproximados que se muestran en los siguientes gráficos independientemente del dispositivo usado en estas pruebas donde se nos muestra el tiempo de respuesta en ms.

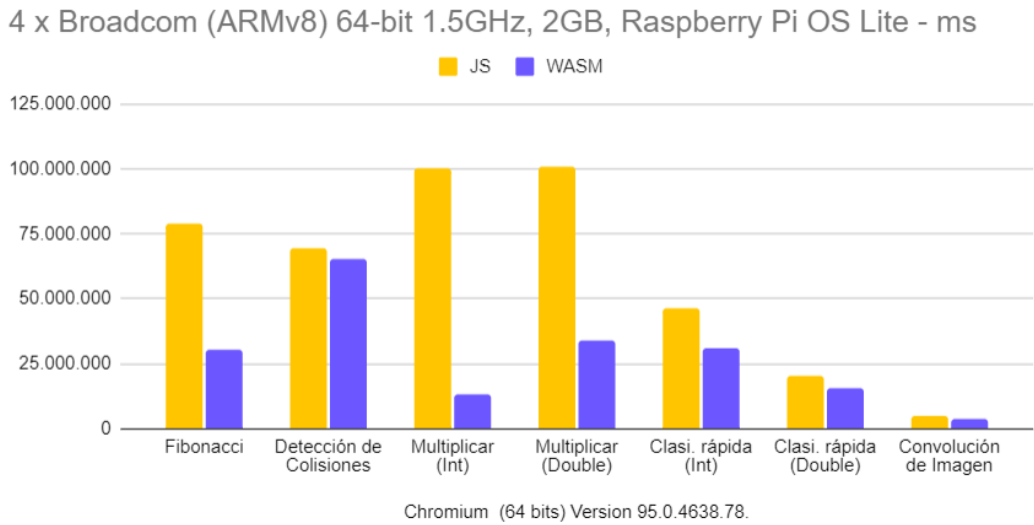
3.1 Windows 10 tiempo de respuesta en ms



3.2 Android 10 tiempo de respuesta en ms

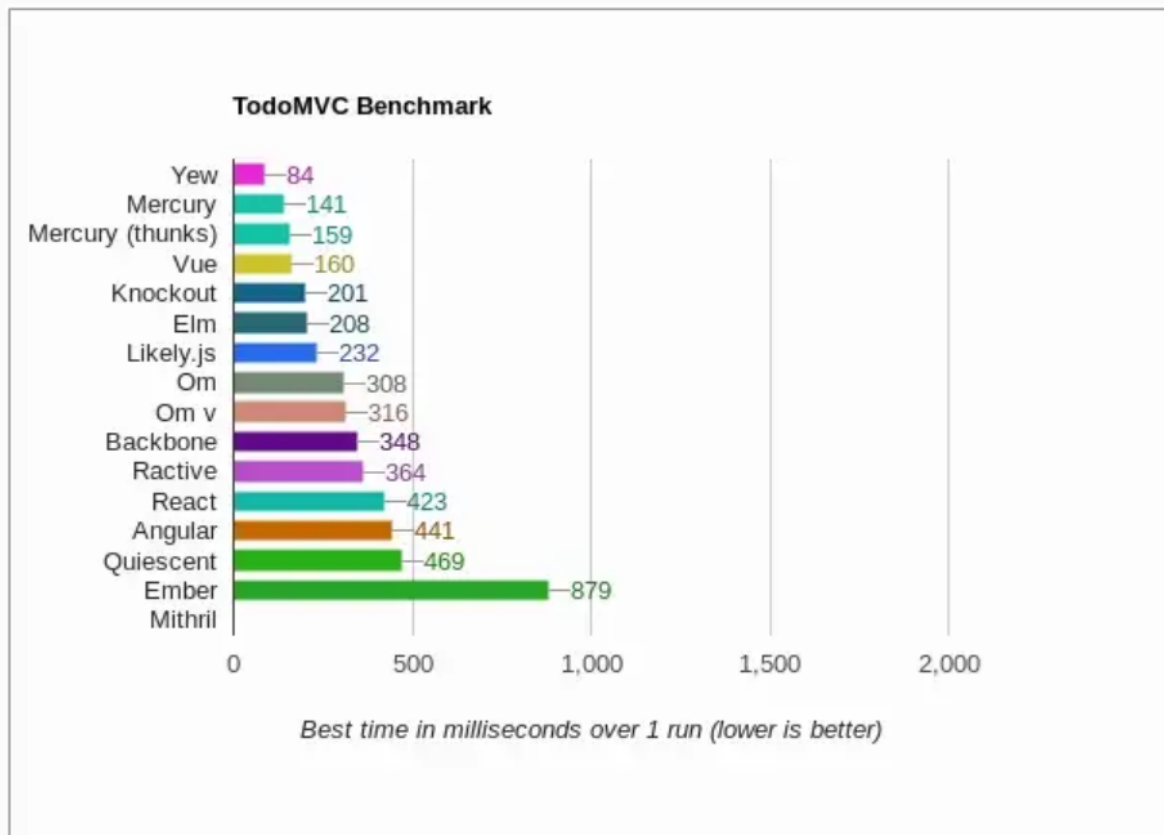


3.3 Raspberry Pi OS tiempo de respuesta en ms



3.4 Yew Frente a otros Frameworks

Segun TodoMVC Benchmark [10] Yew obtiene latencias mas bajas que cualquier otro framework



<https://github.com/DenisKolodin/todomvc-perf-comparación>

4. Conclusiones

Como ya vimos WebAssembly y el Framework Yew que usa Rust obtienen un rendimiento excelente. Todos los navegadores que incorporan la tecnología WebAssembly muestran respuestas más rápidas hasta un 30% más con respecto a Javascript que resultan impresionantes como ya lo vimos con los navegadores de Google Chrome. Por lo tanto, parecería que WebAssembly es muy importante para el futuro de los cálculos numéricos eficientes en la web y aplicaciones complejas por otra parte también haciendo uso del Framework de Yew para el lado del frontend se obtienen resultados muy interesantes no solo por la velocidad si no también por cómo en un futuro no muy lejano ya

podremos empezar a usar otro lenguaje como Rust o distintos para desarrollar una web más eficiente, segura y veloz.

Agradecimiento

A Dios, por brindarme la oportunidad de permitirme disfrutar cada momento de mi vida con tantas cosas maravillosas por descubrir.

A mi familia entera por el apoyo inmenso que me han estado brindando.

A mis compañeros que son parte del equipo Yimi, Jonatan y Grobert por el apoyo brindado durante muchos otros proyectos de universidad.

A mis maestros que se esforzaron por compartir conmigo sus conocimientos y lograr plantar esa semilla de la curiosidad por seguir descubriendo nuevas cosas con respecto a mi carrera.

Referencias Bibliográficas

- [1] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the Web Up to Speed with WebAssembly. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017), 2017.

- [2] J.F. Bastien. WebAssembly – Going public launch bug.<https://github.com/WebAssembly/design/issues/150>, 2015.

- [3] Luke Wagner. WebAssembly consensus and end of Browser Preview. <https://lists.w3.org/Archives/Public/publicwebassembly/2017Feb/0002.html>, 2017.

- [4] Faiz Khan, Vincent Foley-Bourgon, Sujay Kathrotia, Erick Lavoie, and Laurie J. Hendren. Using JavaScript and WebCL for numerical computations: a comparative study of native and

web technologies. In DLS'14, Proceedings of the 10th ACM Symposium on Dynamic Languages, part of SLASH 2014, Portland, OR, USA, October 20-24, 2014, pages 91–102, 2014.

[5] Daniel Lehmann, University of Stuttgart; Johannes Kinder, Bundeswehr University Munich; Michael Pradel. Everything Old is New Again: Binary Security of WebAssembly. Preview. <https://www.usenix.org/system/files/sec20-lehmann.pdf>, 2020

[6] takahirox. WebAssembly-benchmark
<https://github.com/takahirox/WebAssembly-benchmark/tree/master/tests>, 2017

[7] Frank wilson. wasm-js-bm <https://frankmwilsonm.github.io/wasm-js-bm>, 2021

[8] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with WebAssembly. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 185–200. ACM, 2017.

[9] Alon Zakai. Emscripten: an LLVM-to-JavaScript compiler. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, pages 301–312. ACM, 2011.

[10] TodoMVC Benchmark. URL: <https://github.com/DenisKolodin/todomvc-perf-comparison>

[11] YEW Docs. URL: <https://yew.rs/>