ANÁLISIS DEL RENDIMIENTO DE APLICACIONES WEB DESARROLLADAS SOBRE BLAZOR DE .NET CORE, UNA REVISIÓN DE SU ESTADO DEL ARTE

Performance analysis of web applications developed on Blazor from .Net Core, a review of its state of the art

JOHAN ESTIVEN ROBAYO LINARES[1], SANTIAGO ORTIZ SUAREZ[2], ROGER CALDERÓN MORENO[3]

Recibido:02 de febrero de 2024. Aceptado:26 de febrero de 2024

DOI: https://doi.org/10.21017/rimci.1074

Resumen

El presente artículo tiene como objetivo brindar información acerca de los enfoques y características de las aplicaciones web que se pueden desarrollar utilizando el framework Blazor, además se hizo una revisión literaria en la cual comparan diferentes métricas de desempeño de renderizado y comparación de características generales de Blazor con otros frameworks basados en Javascript como lo son Angular, Svelte, Vue, Ember y React, en los casos de estudio mencionados anteriormente se obtuvo que a pesar de que Blazor es relativamente nuevo tiene una gran comunidad que le da soporte en cuanto a librerías, proyectos en GitHub, foros de Stack Overflow, entre otros. Por otro lado, se encontró que en los estudios que compararon rendimiento, Blazor es superior en dos métricas a lo sumo, sin embargo, a pesar de esto, este framework se mantiene estable en la mayoría de las métricas de rendimiento.

Palabras clave: blazor; métricas de desempeño; rendimiento; react; vue; svelte; angular; ember.

Abstract

This article aims to provide insights into the approaches and features of web applications that can be developed using the Blazor framework. Additionally, a literature review was conducted comparing various performance metrics of rendering and general characteristics of Blazor with other JavaScript-based frameworks such as Angular, Svelte, Vue, Ember, and React. In the previously mentioned case studies, it was found that although Blazor is relatively new, it boasts a robust community that supports it through libraries, projects on GitHub, Stack Overflow forums, among others. On the other hand, it was observed in the studies comparing performance that Blazor excels in at most two metrics; however, despite this, the framework remains stable across the majority of performance metrics.

Keywords: blazor; performance metrics; performance; react; vue; svelte; angular; ember.

I. Introducción

En la era digital actual, el desarrollo de aplicaciones web eficientes y dinámicas se ha convertido en un pilar fundamental para la innovación y la excelencia tecnológica. En este contexto, las Single-Page Applications (SPAs) han surgido como una arquitectura vanguardista que redefine la experiencia del usuario al proporcionar interactividad fluida y tiempos de carga optimizados. En este artículo, dirigimos nuestra atención hacia las SPAs, explorando su implementación respaldada por Blazor de .Net Core, una tecnología que ha ganado popularidad significativa en el desarrollo de aplicaciones web.

1 Estudiante de último semestre del programa de Ingeniería de Sistemas. Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. ORCID: https://orcid.org/0009-0003-9073-949X Correo electrónico: johan.robayo@unillanos.edu.co

2 Estudiante de último semestre del programa de Ingeniería de Sistemas. Profesor Asistente Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. ORCID: https://orcid.org/0009-0002-3373-0061 Correo electrónico: santiago.ortiz.suarez@unillanos.edu.co

3 Ingeniero de Sistemas, Especialista en Ingeniería de Software, Magister en Software Libre. Profesor Asistente Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. ORCID: https://orcid.org/0000-0001-5923-8601 Correo electrónico: rcalderonmoreno@unillanos.edu.co

El uso de Blazor en el desarrollo de SPAs ofrece una perspectiva intrigante, fusionando la versatilidad de .Net Core con la potencia de las SPAs[1]. A medida que la comunidad de desarrolladores busca adoptar tecnologías que mejoren la eficiencia y la calidad del software, es imperativo realizar una exhaustiva revisión del estado del arte de las aplicaciones web basadas en Blazor. Por lo tanto, se aborda esta necesidad, evaluando las aplicaciones web desde diversas perspectivas, como rendimiento, mantenimiento, escalabilidad, velocidad y facilidad de desarrollo, para determinar su capacidad como tecnología de base en el desarrollo de software.

A través de una revisión literaria, se busca proporcionar una visión integral del panorama actual de las SPAs respaldadas por Blazor, destacando sus fortalezas y consideraciones clave. Al hacerlo, se brinda un recurso valioso que facilite una toma de decisiones informada al considerar la adopción de Blazor en el desarrollo de aplicaciones web como tecnología base para el desarrollo de software.

II. Desarrollo del tema. Definiciones

En este artículo de revisión literaria, es importante comprender en primera instancia los conceptos generales de (SPAs) soportadas por Blazor de .Net Core como base para desarrollar software.

A. Single-Page Application (SPAs)

Es una aplicación web que carga una sola página Html y actualiza su contenido dinámicamente, proporcionando una experiencia de usuario más rápida y fluida al evitar recargar la página completa con cada interacción.

Algunas de sus principales ventajas son: experiencia de usuario fluida, interactividad y velocidad, fácil mantenimiento, compatibilidad con dispositivos móviles y mejora del rendimiento[2].

B. ¿Qué es Blazor?

Es un framework orientado a la construcción de interfaces web altamente interactivas para los usuarios en el lado del cliente, lo cual implica representar las interfaces de usuario mediante HTML y CSS para garantizar su compatibilidad con diversos navegadores web y dispositivos móviles. A diferencia de la mayoría de los frameworks web del lado del cliente que se basan en JavaScript, Blazor ofrece la posibilidad de crear estas interfaces de usuario utilizando el lenguaje de programación C#[3].

Blazor destaca por permitir el desarrollo de aplicaciones web interactivas gracias a el lenguaje de programación C#, lo que facilita la reutilización de código entre el frontend y el backend. Su enfoque en Single-Page Applications (SPAs) mejora la experiencia del usuario, mientras que la integración con .NET proporciona acceso a herramientas y bibliotecas existentes. Gracias a WebAssembly, Blazor ofrece un rendimiento cercano al código nativo y su modelo de programación basado en componentes simplifica el manejo declarativo de la interfaz de usuario. La compatibilidad con diversos navegadores, el soporte de herramientas populares como Visual Studio, y una comunidad activa hacen de Blazor una opción atractiva para el desarrollo eficiente y moderno de aplicaciones web.

C. Modelos de hospedaje de Blazor

1. **Blazor Server:** Es un enfoque de hospedaje de componentes Razor en el servidor de una aplicación ASP.NET Core, donde el servidor maneja la ejecución del código C#, el envío de eventos de interfaz de usuario desde el navegador al servidor y la aplicación de actualizaciones de interfaz de usuario. Un componente Razor es una manera de construir vistas reutilizables que pueden ser usadas en cualquier parte de una aplicación Blazor[4]. Este modelo de hospedaje utiliza una conexión de SignalR, una biblioteca de comunicación en tiempo real de código abierto que facilita la conexión entre clientes y servidores. Gracias a su versatilidad, SignalR selecciona automáticamente el protocolo más apropiado en función de las capacidades individuales de los servidores y clientes. Además, esta conexión también se utiliza para gestionar las llamadas de interoperabilidad de Javascript[5]. La comunicación entre Blazor server y el cliente (cualquier navegador) se ilustra en la Fig. 1.

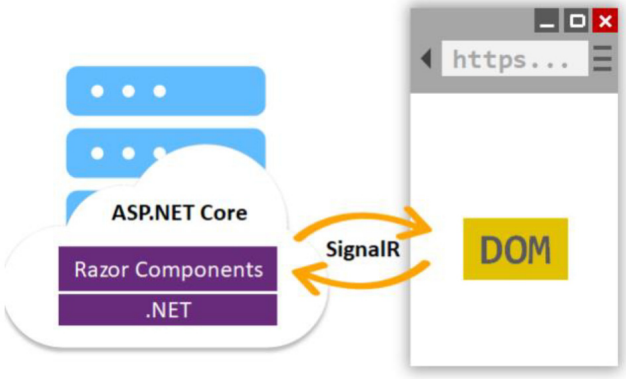

Fig. 1. Modelo de Blazor Server, tomado del sitio oficial de Microsoft[5].

2. **Blazor Webassembly:** Se trata de una estrategia de alojamiento diseñada para aplicaciones web de página única, conocidas como SPAs por sus siglas en inglés. Su objetivo es compilar aplicaciones interactivas del lado del cliente utilizando .NET. La ejecución de código .NET en navegadores web, se realiza mediante el estándar WebAssembly (Wasm). WebAssembly es un formato de código binario eficiente que garantiza descargas rápidas y una ejecución óptima. Además, es compatible con todos los navegadores web modernos, incluyendo aquellos diseñados para dispositivos móviles[6]. La fig. 2 muestra la estructura de un proyecto de Blazor WebAssembly.

Esta combinación de tecnologías refuerza Blazor WebAssembly como una opción atractiva para desarrollos web dinámicos y eficaces.

3. **Blazor Hybrid:** Es una tecnología basada del framework .NET que aprovecha tecnologías web para su funcionalidad, implementando componentes de Razor que se ejecutan directamente en la aplicación nativa. A diferencia de la ejecución en WebAssembly, estos componentes coexisten con otros códigos de .NET en la aplicación. En este contexto, los componentes representan la interfaz de usuario web, basada en HTML y CSS, dentro de un control Webview integrado mediante un canal de interoperabilidad local[7]. La fig. 3 ilustra la manera en la cual este modelo de hospedaje utiliza los componentes Razor en plataformas nativas.

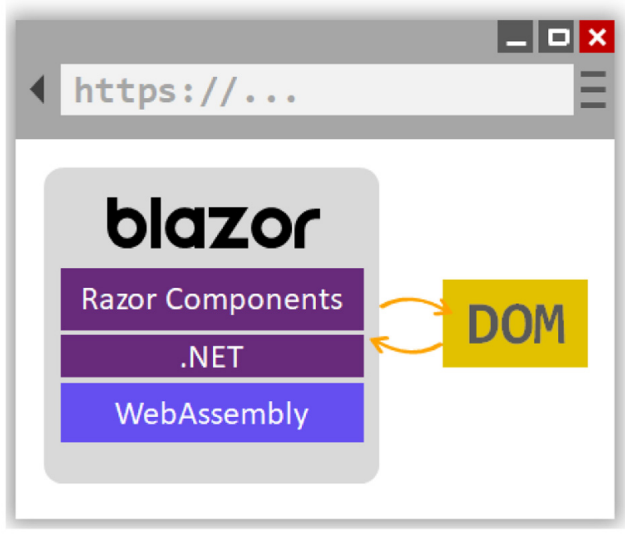

Fig. 2. Modelo de Blazor WebAssembly, tomado del sitio oficial de Microsoft[6].

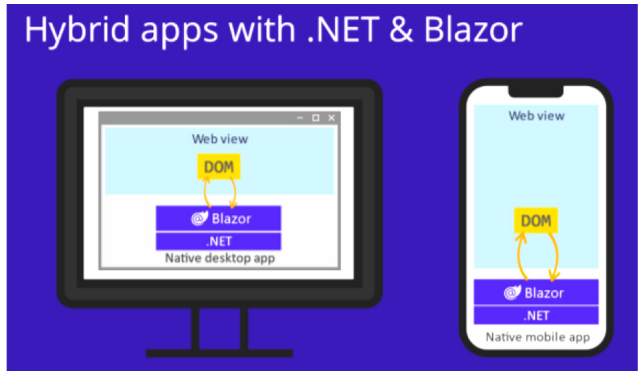

Fig. 3. Modelo de Aplicaciones híbridas con .NET & Blazor[8], tomado del sitio oficial de Microsoft.

Por lo tanto, Blazor Hybrid ofrece a los desarrolladores una herramienta poderosa y flexible para la creación de aplicaciones multiplataforma, aprovechando las ventajas de los lenguajes de programación más populares y la capacidad de ejecutar componentes en diferentes entornos.

D. Componentes De Blazor

Blazor, al igual que muchos frameworks frontend modernos, permite crear aplicaciones web basadas en componentes. Un componente se define como un elemento de la interfaz de usuario como una página, un cuadro de diálogo u otro. Además, cada componente define la lógica para la representación de la interfaz de usuario mediante el uso de eventos para controlar las acciones de los usuarios[9].

En el ecosistema de Blazor, los componentes se identifican por archivos. Razor. La combinación de HTML y C# en el lenguaje de plantillas Razor permite definir la lógica de interacción de manera clara. Cada componente Razor se compila en una clase C# en .NET, lo que simplifica el desarrollo y potencia la coherencia del código, estos componentes Razor tienen las siguientes características:

- Definen la lógica.
- Controlan acciones del usuario.
- Se pueden reutilizar y anidar.
- Se pueden compartir como bibliotecas o paquetes Nuget. (útil para proveer componentes en aplicaciones multiplataforma).

E. Integración de Asp.Net Core con Blazor

La integración de las herramientas proporcionadas por ASP.NET Core con Blazor ofrecen una poderosa sinergia para el desarrollo web moderno. ASP.NET Core, como un marco de trabajo de servidor web multiplataforma y de código abierto, se convierte en la base robusta para la construcción de aplicaciones escalables y eficientes. Su capacidad para gestionar la lógica del servidor, manejar solicitudes HTTP y proporcionar una arquitectura modular hace que sea una elección ideal para trabajar en conjunto con Blazor.

Por otro lado, lo que hace única a esta combinación es la capacidad de Blazor para permitir el desarrollo de aplicaciones de una sola página (SPA) utilizando C# en lugar de Javascript, lo que facilita la creación de aplicaciones web modernas con el mismo lenguaje de programación en ambos lados: cliente y servidor. La integración efectiva de ASP.NET Core con Blazor no solo simplifica el flujo de trabajo del desarrollador al proporcionar una coherencia en el uso del lenguaje, sino que también maximiza la reutilización del código y fomenta la creación de aplicaciones web rápidas, seguras y de alto rendimiento[10].

III. Estado del arte

Con el objetivo de contextualizar el presente artículo, se procederá a explorar trabajos, tesis de grado, proyectos o artículos científicos relacionados con el enfoque de Single-Page Application (SPA) respaldado por Blazor de .NET Core. Esta revisión del estado del arte tiene como finalidad evaluar la idoneidad de Blazor como tecnología base para el desarrollo de software centrado en aplicaciones web.

1. Tomado de: Evaluación de Blazor WebAssembly para una aplicación web progresiva (PWA)

Esta tesis fue elaborada en Suecia en el año 2012[10]. En este estudio, se lleva a cabo una evaluación del marco Blazor WebAssembly en el contexto de la metodología de Aplicación Web Progresiva (PWA). Se realizó una comparación con una PWA de ReactJS como referencia, centrándose en el rendimiento inicial y la documentación relacionada con PWA. Para el análisis del rendimiento, se desarrolló una PWA tanto en Blazor WebAssembly como en ReactJs.

El experimento que se llevó a cabo para medir el rendimiento se dividió en 3 sesiones con un total de 5392 puntos de datos, donde las 3 mediciones de ReactJS proporcionaron 900 puntos cada una, mientras que las mediciones de Blazor Wasm variaron entre 873 y 930, como se evidencia en la Tabla I:

Tabla I. Sesiones del experimento y cantidad máxima de elementos[10]

Sample	Session 1 points	Session 2 points	Session 3 points
Blazor Wasm	873	889	930
ReactJS	900	900	900

Especificaciones. El experimento se llevó a cabo en una computadora personal con las siguientes características:

- Sistema operativo-Windows 10 Edición Educativa de 64 bits
- CPU - AMD A10 PRO-7350B R6, 10 Compute Cores 4C + 6G (4 CPUs), ~2,1GHz
- RAM - 8 GB (6,95 GB Usable)

Análisis y resultados:

En este estudio se plantearon las siguientes preguntas:

1. ¿Cuánto tiempo lleva generar un árbol VDOM entre los dos frameworks?

2. ¿Cuál es la diferencia de tamaño de los árboles VDOM en la memoria entre Blazor Wasm y ReactJS?

3. ¿Qué tan consistente es cada implementación de VDOM en tiempos de generación?

Con respecto a la primera pregunta, la aplicación PWA de ReactJS tuvo los tiempos de generación del virtual DOM (VDOM) más largos para las 3 sesiones y sus tiempos de carga aumentaron mucho más rápido que los tiempos de carga de Blazor Wasm, como se observa en la Fig. 4.

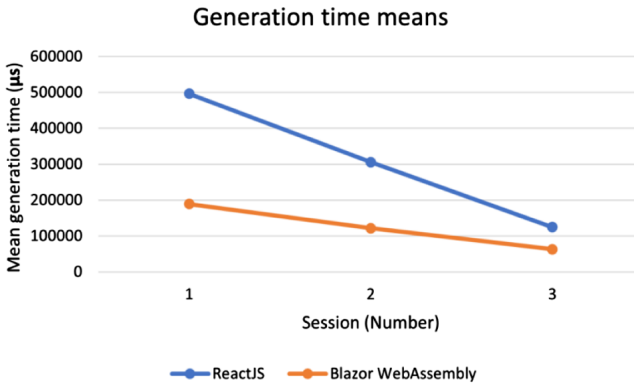

Fig. 4. Tiempo de generación del VDOM para ReactJS y Blazor WebAssembly[10]

Adicionalmente, se observó una disminución de alrededor del 61,9%, 60,1% y 49,3% en los tiempos de generación en Blazor Wasm. Estos datos están en el rango de mediciones previas en estudios que se han realizado anteriormente por Reiser & Bläser[11], Haas et al[12] y Wang et al[13], en donde se menciona que el estándar WebAssembly es un factor en los tiempos de generación más bajo entre ReactJS y Blazor Wasm.

Respecto a la pregunta 2, se analizó la diferencia de tamaño de los árboles VDOM en la memoria entre Blazor y ReactJS, donde se dió como resultado que Blazor Wasm utiliza menos memoria después de generar los árboles durante cada sesión, en cambio, ReactJS consumió más memoria cuando más elementos del árbol tuvo que manejar. Por ende la diferencia en el tamaño medio del árbol se redujo de la sesión 1 a la sesión 3, como se evidencia en la Fig. 5.

Fig. 5. Tamaño medio del árbol para cada framework[10].

En esta sección se demostró que la aplicación PWA Blazor Wasm en comparación con la PWA de ReactJS utilizó 0,34 MB, 0,25 MB, Y 0,18 MB menos memoria media al generar árboles VDOM durante cada sesión.

Por último, para determinar la consistencia de cada implementación de VDOM en tiempos de generación, se definió la consistencia como la desviación estándar, ya que cuanto menor sea la desviación estándar, menor será la dispersión de los datos tomados. A continuación, la figura 6 muestra la desviación estándar calculada entre ambos frameworks durante las 3 sesiones realizadas.

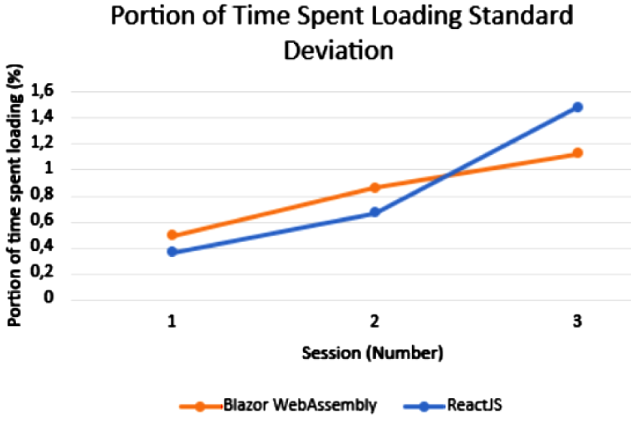

Fig. 6. Porción del tiempo dedicado a cargar la desviación estándar[10].

Según los datos de la Fig. 6, se determinó que Blazor Wasm es más inconsistente en comparación con ReactJS cuando maneja muchos elementos, ya que a pesar de que en las sesiones 1 y 2 donde se tomaron 873 y 889 puntos respectivamente para Blazor Wasm, este framework obtuvo una mayor desviación estándar. Por otro lado, para ReactJS se tomaron 900 datos para las 3 sesiones, a pesar de que se tomaron más datos en ReactJS que en Blazor Wasm, este framework obtuvo una menor desviación estándar, lo cual lo hace más consistente en cada implementación de VDOM en tiempos de generación.

Por otro lado, se destaca que la documentación sobre la terminología de aplicaciones web progresivas difiere entre los dos marcos. La documentación de Blazor WebAssembly abarca más temas y proporciona conocimiento de primera mano, mientras que la documentación de ReactJS cubre menos temas y se basa en fuentes externas para explicaciones.

En conclusión, los resultados y el análisis revelan que Blazor WebAssembly y ReactJS presentan diferencias significativas en cuanto a la calidad y contenido de sus documentaciones. Asimismo, se destaca que, si bien Blazor WebAssembly con VDOM ofrece tiempos de carga más rápidos, se observa cierta inconsistencia en comparación con la estabilidad de ReactJS. Estos hallazgos subrayan la importancia de considerar no solo el rendimiento sino también la coherencia en la elección entre Blazor WebAssembly y ReactJS, dependiendo de las prioridades y requisitos específicos del desarrollo de aplicaciones web progresivas.

2. Tomado de: Evaluación de los frameworks Blazor y Angular. Rendimiento para aplicaciones web.

Esta tesis fue elaborada en Suecia; en la Universidad de Dalarna en el año 2020[14], el objetivo de este caso de estudio fue destacar la evolución de la programación web desde la introducción de JavaScript en la década de 1990 hasta la aparición de nuevos marcos de desarrollo, como Angular y, más recientemente, Blazor de Microsoft. Se señala la falta de comparaciones de rendimiento para Blazor y la necesidad de evaluar su capacidad frente a uno de los principales marcos, Angular, especialmente dado el enfoque de Blazor en prescindir de JavaScript. El autor destaca el lanzamiento de Blazor en mayo de 2020 y plantea la pregunta de si este marco puede competir en rendimiento con los marcos tradicionales basados en JavaScript. El propósito del estudio fue realizar una evaluación imparcial del rendimiento de Blazor y Angular, utilizando dos aplicaciones web menores con el mismo código de backend y operaciones CRUD. La medición del rendimiento se realiza mediante la herramienta «Lighthouse» de Google, con el objetivo de proporcionar información valiosa para desarrolladores y empresas que consideren alejarse de JavaScript y adoptar Blazor. Las métricas para la evaluación del desempeño de cada framework fueron las siguientes:

- **First Contentful Paint.** Mide cuánto tiempo tarda el navegador en representar la primera parte del contenido DOM(Document Object Model), por ejemplo: texto, imágenes, después de que un usuario navega a una determinada página.
- **Time to interactive.** Esta métrica mide el tiempo que tarda la página en convertirse completamente interactiva.
- **Speed index**. Muestra la rapidez con la que se muestran visualmente los contenidos de una página durante la carga de la página.
- **Largest Contentful Paint.** Marca el momento en el que se muestra en la pantalla el texto o la imagen más grande.
- **Total Blocking Time.** Mide la cantidad total de tiempo que una página no puede responder a la entrada del usuario, como clics del mouse, toques en la pantalla o pulsaciones del teclado.
- **Cumulative Layout Shift.** Mide el movimiento inesperado del contenido de la página. Se produce un cambio de diseño cada vez que un elemento visible cambia su posición de un cuadro renderizado al siguiente.

Análisis y resultados

Para este estudio se midieron veinte pruebas de rendimiento con cada framework, donde el promedio de cada medición de cada métrica se calculó y se graficó respectivamente.

La Fig. 7 muestra la métrica de rendimiento First contentful Paint, se puede observar que durante las 20 mediciones tomadas, Blazor mostró un rendimiento estable de 0,4 segundos, mientras que Angular mostró un rendimiento variable entre los 0,4 segundos y 4,3 segundos. Esto demuestra que Blazor fue superior mostrando la primera parte del contenido DOM.

Como se observa en la Fig. 8, nuevamente es notable que Blazor muestra un rendimiento muy estable de 7,8 segundos, mientras que Angular varía desde 1,7 hasta 9,6 segundos. A pesar de que Angular obtuvo un desempeño irregular con tiempos variados, Angular en promedio puede tardar menos tiempo en hacer que una página se vuelva completamente interactiva.

En la Fig. 9 se muestra que Blazor se mantiene estable entre 0,9 y 1,2 segundos, Por otra parte, Angular mostró un resultado inestable entre 0,6 y 4,7 segundos. Además, Blazor obtuvo un índice de velocidad promedio menor de 0,93 segundos, y Angular obtuvo un índice de velocidad promedio de 2,88 segundos.

Como se evidencia en la Fig. 10, los dos frameworks se mantuvieron estables, sin embargo, Blazor osciló entre 270 y 340 milisegundos y Angular osciló entre 40 y 90 milisegundos, lo cual indica que Angular tuvo mejor desempeño en esta métrica.

La Fig. 11 muestra que Blazor nuevamente obtuvo un resultado consistente entre los 12 y 12,2 segundos. Por otro lado, Angular osciló entre 1,6 y 12,7 segundos durante las veinte pruebas. En

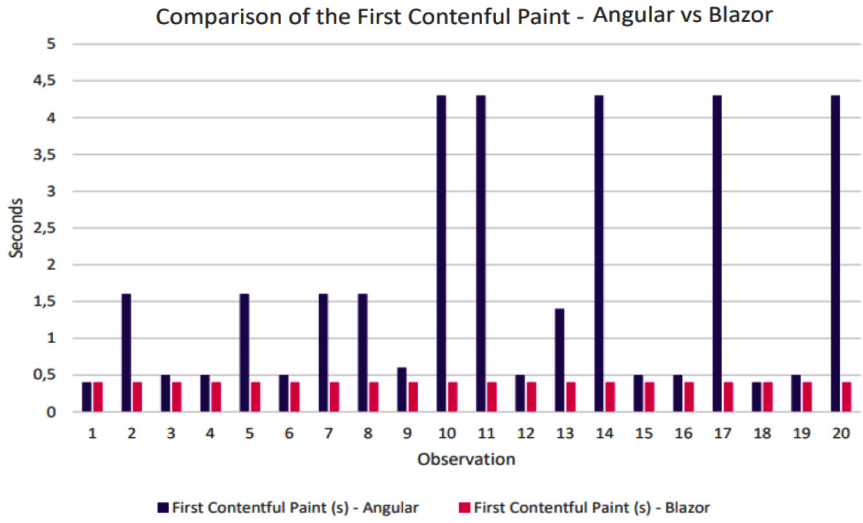

Fig. 7. Comparación de la métrica first contentful paint entre Blazor y Angular[14].

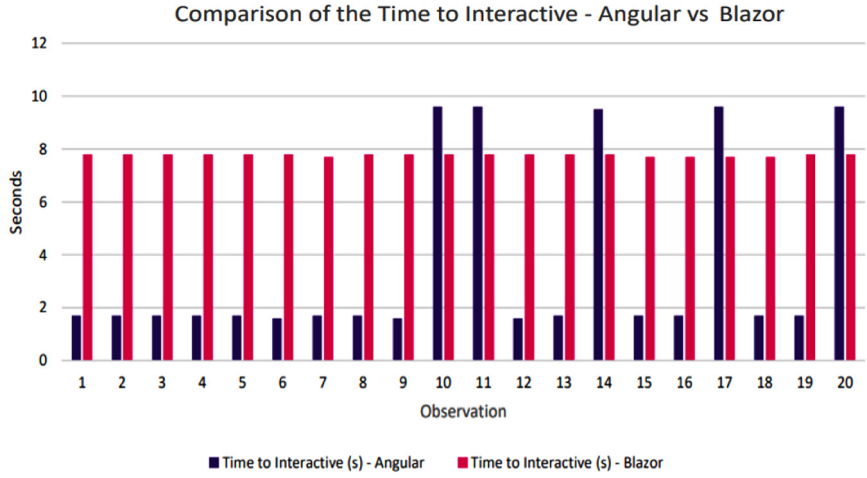

Fig. 8. Comparación de la métrica Time to interactive entre Blazor y Angular[14].

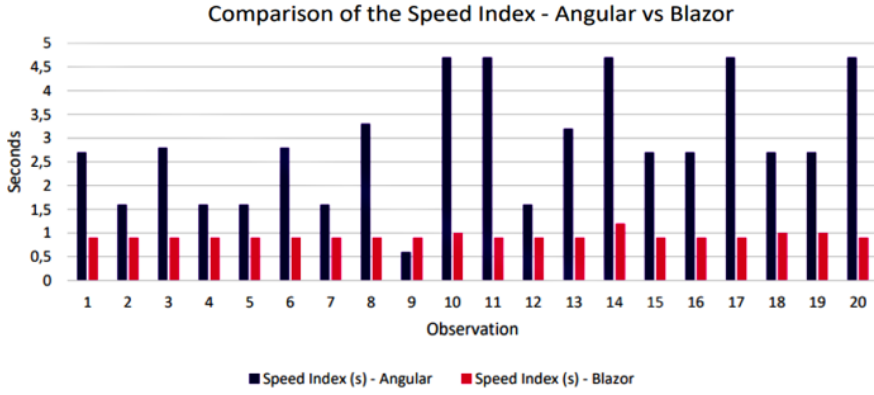

Fig. 9. Comparación de la métrica Speed Index entre Angular y Blazor[14]

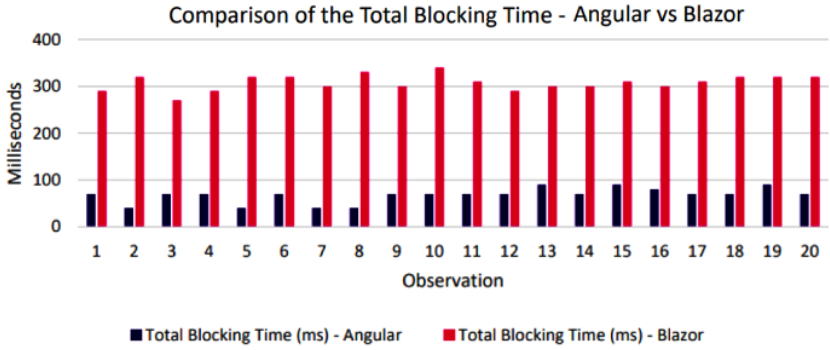

Fig. 10. Comparación de la métrica total blocking time entre Blazor y Angular[14].

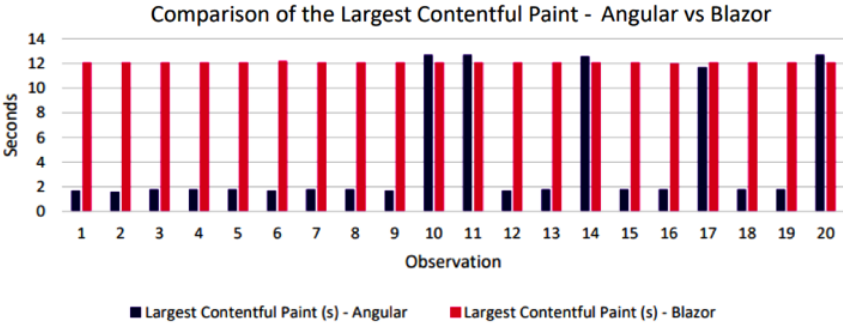

Fig. 11. Comparación para la métrica Largest Contentful Paint entre Angular y Blazor[14]

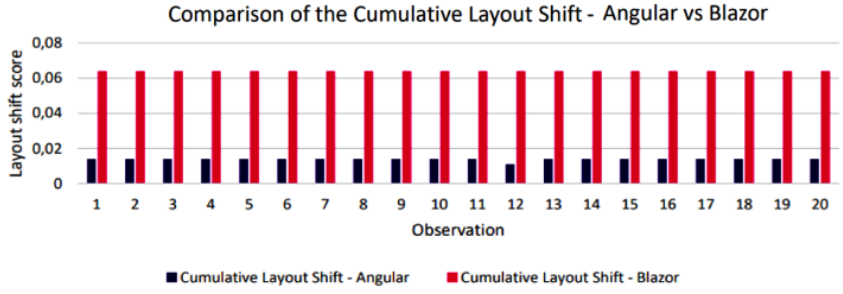

Fig. 12. Comparación de la métrica Cumulative Layout Shift entre Angular y Blazor[14].

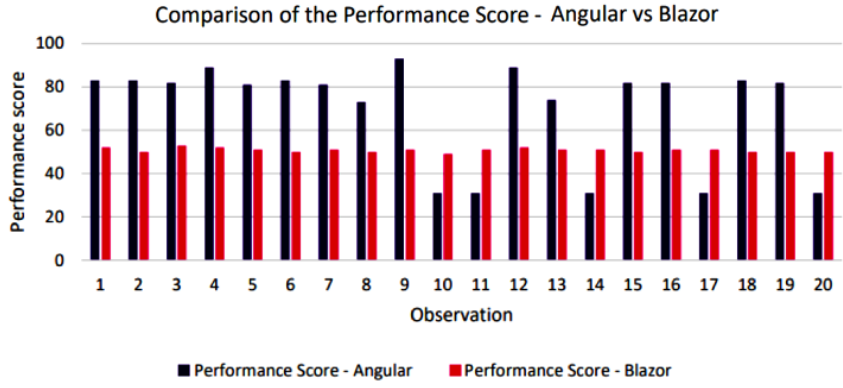

Fig. 13. Comparación de la puntuación de rendimiento entre Angular y Blazor[14].

cuanto al tiempo promedio para esta métrica, Angular tuvo variación en los datos, sin embargo, presentó un promedio de 4,44 segundos y Blazor un promedio de 12,1 segundos, lo cual indica que Angular tardó menos tiempo en pintar el texto o la imagen más grande para el usuario.

En la Fig. 12 se muestra la métrica Cumulative Layout Shift, donde una puntuación inferior a 0,1 se considera buena, entre 0,1 y 0,25 significa que necesita mejorar y cualquier puntuación superior a 0,25 se considera mala para la experiencia del usuario. Ambos frameworks obtuvieron una puntuación por debajo de 0,1 Angular obtuvo un valor de 0,014 y Blazor obtuvo un valor de 0,064.

La Fig.13 muestra la puntuación de rendimiento total teniendo en cuenta las seis métricas que se realizaron en las 20 pruebas. Esta puntuación fue calculada por el software Lighthouse, donde un mayor valor numérico determina una mejor puntuación del rendimiento. Cabe destacar que Blazor tuvo una puntuación de rendimiento estable con un pequeño rango de 49 a 53 y Angular osciló entre 31 y 93.

El análisis de la tesis indica que Angular superó a Blazor en cuatro de los seis criterios de rendimiento evaluados, según la Tabla II. Aunque Angular mostró un rendimiento generalmente mejor, la evaluación evidenció inconsistencias en sus resultados, a diferencia de la consistencia y estabilidad de Blazor durante las pruebas. La posibilidad de mejorar el rendimiento de Blazor, incluso superando a Angular tras la publicación de la aplicación, sugiere su capacidad para

Tabla II. Resultados finales de la prueba[14].

Criterion	Framework Evaluation favors
First Contentful Paint	Blazor
Time to Interactive	Angular
Speed Index	Blazor
Total Blocking Time	Angular
Largest Contentful Paint	Angular
Cumulative Layout Shift	Angular
Performance Score	Angular

competir como un marco de desarrollo web moderno. A pesar de su rendimiento inferior en la evaluación, Blazor se destaca como una opción favorable para nuevos proyectos, especialmente para aquellos con poca experiencia en desarrollo web o familiaridad con C# u otros lenguajes de programación similares.

3. Tomado de: Evaluación del framework Blazor, una comparación entre Blazor y React.

Esta tesis se ha realizado en la Escuela de Ingeniería de la Universidad de Jönköping en Suecia dentro de Ingeniería Informática[15]. La tesis se centra en determinar si Blazor representa una opción viable para el desarrollo de aplicaciones web y si se posiciona como un framework sólidamente establecido en comparación con los frameworks de JavaScript. Con el propósito de evaluar la idoneidad de Blazor, se llevó a cabo una comparación con React, abordando diversos criterios tales como líneas de código, complejidad ciclomática, gestión del estado, herramientas de depuración, comunidad y librerías. Para respaldar esta evaluación, se desarrollaron dos aplicaciones en cada framework, y se examinó la documentación correspondiente. La motivación detrás de esta investigación radica en la escasez de estudios exhaustivos que comparen Blazor con marcos establecidos como Angular, React y Vue.

Análisis y resultados

En términos de líneas de código, Blazor presenta componentes más compactos que React, siendo hasta un 50% más pequeños en algunos casos. La complejidad ciclomática de los componentes es similar entre ambos, mientras que en la gestión estatal, se destaca que la biblioteca Fluxor en Blazor puede lograr resultados equiparables a Redux.

En cuanto a bibliotecas, se compararon varias, destacando que Blazor AntDesign y MatBlazor tienen menor popularidad que sus contrapartes en React.

A nivel de comunidad, React muestra una presencia significativamente mayor en Stack Overflow, GitHub (en términos de repositorios, contribuyentes y estrellas) que Blazor.

En cuanto al ecosistema, React se destaca como un marco sólidamente establecido con una amplia comunidad y abundantes bibliotecas, mientras que Blazor, aunque no ha alcanzado ese nivel, muestra un crecimiento prometedor. En conclusión, Blazor se posiciona como una opción válida para el desarrollo de aplicaciones de una sola página, y su ecosistema demuestra estar en desarrollo, considerando su relativa novedad en comparación con React. Se sugieren futuras investigaciones mediante entrevistas con partes interesadas y el seguimiento de la evolución del ecosistema de Blazor para obtener una comprensión más completa de su posición en comparación con React.

Este estudio dió como conclusión que Blazor es una opción válida para el desarrollo de aplicaciones de una sóla página (SPA) y su ecosistema está fuertemente establecido teniendo en cuenta su antigüedad.

4. Tomado de: Frameworks web modernos: Una comparación del rendimiento de renderizado.

Esta tesis final se ha realizado en la Universidad de Helsinki en Finlandia,en el departamento de ciencias de la computación[16].

Esta tesis consiste en comparar las estrategias de renderizado empleadas en Angular, React, Vue, Svelte y Blazor, los frameworks más utilizados en el desarrollo de aplicaciones web. Se revisó cómo funcionan sus estrategias de renderizado y se presentó una forma de estimar los niveles de rendimiento. Por otra parte, se construyó una aplicación con idénticas características en cada uno de estos frameworks, seguido por la medición del tiempo requerido para ejecutar el script durante el ciclo de renderizado. Se llevaron a cabo un total de 5 escenarios, cada uno evaluando aspectos específicos del renderizado al variar el tamaño y la estructura del árbol de componentes. Cada prueba

se repitió en 10 ocasiones y los resultados se presentan como el promedio obtenido de estas 10 iteraciones. Las pruebas realizadas se hicieron en una computadora de escritorio con una CPU Ryzen 5 3600 y 16 GB de RAM.

Análisis y resultados

Los resultados de los puntos de referencia proporcionan información valiosa sobre el rendimiento de los marcos web modernos en diversas situaciones. En el primer escenario, centrado en la creación de elementos estáticos, se observa que Angular muestra un rendimiento superior a los demás, seguido de cerca por React y Svelte. En el segundo escenario, que evalúa el costo de crear componentes en forma de árbol binario, Angular nuevamente destaca en eficiencia, seguido de React y Svelte. Cuando se trata de la actualización del componente raíz en un árbol de componentes, React y Svelte exhiben tiempos de ejecución notables, mientras que Angular y Blazor muestran un rendimiento aceptable. Similarmente, al actualizar un componente hoja, React y Svelte destacan, superando a los demás marcos.

En el escenario final, que mide el costo de actualizar componentes con contenido estático a dinámico, React y Svelte demuestran un rendimiento impresionante, especialmente en comparación con Angular y Blazor. Estos resultados sugieren que la elección del marco web puede depender de las necesidades específicas de la aplicación, ya que cada marco muestra fortalezas y debilidades en diferentes contextos. Estos hallazgos proporcionan una visión detallada sobre cómo los frameworks web seleccionados abordan los desafíos de rendimiento en la creación y actualización de elementos y componentes en diversas configuraciones.

En términos específicos de los marcos web evaluados, se destaca que Blazor, el único basado en WebAssembly, muestra un rendimiento significativamente inferior en comparación con sus contrapartes basadas en JavaScript, como React. La mediación del acceso a las API DOM a través de una capa de JavaScript se identifica como una posible causa de la sobrecarga observada en Blazor, aunque no se puede determinar si esto se debe a factores específicos de Blazor o a una limitación inherente de WebAssembly.

Se enfatiza la importancia de los sistemas de reactividad para detectar automáticamente componentes, siendo Svelte el marco más destacado en este aspecto al exhibir un rendimiento consistente y superior en todos los puntos de referencia.

En resumen, la navegación tradicional de página web se revela inadecuada para aplicaciones interactivas y ricas en contenido. Los marcos web modernos, como Angular, React, Vue, Svelte y Blazor, ofrecen soluciones mediante abstracciones declarativas que reducen la complejidad en la gestión del estado de la interfaz de usuario.

En la Fig. 14 se puede observar que Blazor obtuvo los tiempos más altos a medida que se aumentaban el número de componentes en el árbol. Cabe resaltar que, durante las 5 pruebas realizadas, el framework que presentó un mayor rendimiento fue Svelte independientemente de su estrategia para renderizar los componentes.

Por lo tanto la tesis destaca diferencias cruciales en las estrategias de renderizado de estos marcos, impactando directamente en su rendimiento. La optimización de contenido estático a través de compiladores y la implementación de sistemas reactivos son factores clave para mejorar el rendimiento. En un contexto donde el navegador sigue siendo la principal plataforma de aplicaciones, comprender estas herramientas se vuelve crucial para garantizar una web accesible y eficiente, especialmente en dispositivos con recursos limitados.

5. Tomado de: Sistema de reservación de ticket usando Blazor

Esta tesis fue elaborada en la Universidad Masaryk facultad de informática en el año 2020 en República Checa[17]. La tesis se enfoca en implementar un sistema de reserva de boletos gratuitos para cines más pequeños mediante el uso del marco de trabajo Blazor de Microsoft. Se revisa el estado actual del desarrollo web, comparando los marcos de trabajo JavaScript populares como Vue, React y Angular. La implementación se basa en el modelo de hospedaje de WebAssembly, con una evaluación de los beneficios y desventajas de WebAssembly.

La tesis examina detalladamente el marco de trabajo Blazor, sus internos y características esen-

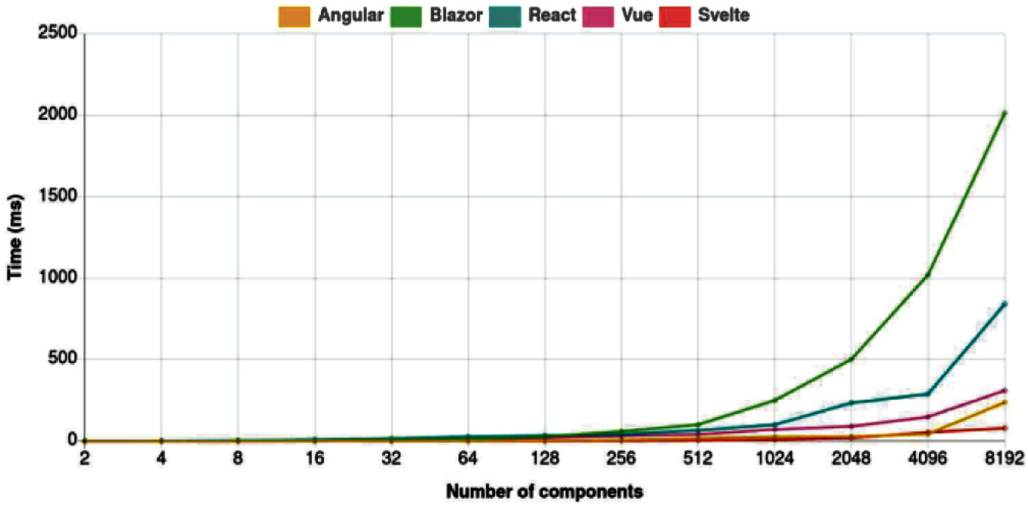

Fig. 14. Tiempos de ejecución al actualizar todos los componentes en un árbol de N componentes donde cada componente contiene principalmente contenido estático[13].

ciales. La implementación se divide en base de datos, backend y frontend, abordando problemas encontrados durante el desarrollo y sus soluciones. Además se realizó una comparación para examinar la popularidad y los factores de rendimiento entre Vue, Angular y Blazor, ya que estos dos factores son importantes para decidir cuál tecnología escoger.

Análisis y resultados

Comparación de popularidad. Según este caso de estudio, el factor de popularidad es importante debido a que indica el tamaño de la comunidad, la cantidad de recursos y materiales encontrados. Para esta comparación, los datos fueron tomados de los sitios web más visitados por los desarrolladores: GitHub, NPM (Node Package Manager), y StackOverflow.

Los datos tomados de GitHub, fueron la cantidad de estrellas de cada proyecto, por parte de NPM, se tomó la cantidad de descargas de cada framework por año, exceptuando a Blazor debido a que no es un framework basado en Javascript. Finalmente, se tomó de la tendencia en StackOverflow de cada framework como un porcentaje, estos datos se presentaron en la Tabla III.

En base a este criterio de evaluación, se determina que React emerge como la elección más destacada al desarrollar una aplicación con un framework de JavaScript. Por otro lado, se destaca que Blazor presenta una comunidad reducida, lo que implica que es el framework menos propenso a contar con una amplia variedad de bibliotecas y soporte al trabajar en el desarrollo de una aplicación.

Tabla III. Popularidad de frameworks[17].

Framework	GitHub Stars	NPM Downloads/Year	SO Trends
React	159,962	8,152,612	4.6%
Vue	176,129	1,893,959	1.25%
Angular	68,386	2,013,743	2.1%
Blazor	9297(20207)	-	0.2%

Comparación de rendimiento. Para medir el rendimiento de los frameworks estudiados, se usó una herramienta llamada Lighthouse, en este caso se midió las siguientes métricas: Rendimiento, Accesibilidad, mejores prácticas y tamaño, los datos tomados para los diferentes frameworks se muestran en la Tabla IV.

En relación a este criterio de evaluación, se determina que el tamaño de la aplicación desarrollada en Blazor es notablemente mayor que el de los frameworks basados en JavaScript. Esto se atribuye al hecho de que las aplicaciones creadas en Blazor WebAssembly descargan archivos necesarios en el navegador junto con el Runtime de

Tabla IV. Rendimiento de los frameworks medidos por Lighthouse[17]

Framework	Performance	Accessibility	Best practices	Size
React	90	64	93	43KB
Vue	91.8	76	93	23KB
Angular	94.2	70	93	143KB
Blazor	93.2	84	100	2500KB

C#, generando así aplicaciones más voluminosas. Sin embargo, cabe destacar que Blazor supera a otros frameworks en términos de capacidad de respuesta, transiciones de página y velocidad general de carga, lo cual también se confirma en la tabla IV[17].

6. Tomado de: Comparación de desarrollo de aplicaciones entre Blazor, React y Angular

Esta tesis final se ha realizado en la Universidad de Mariboru en Eslovenia, programa de tecnologías de la informática y la comunicación[18].

Esta tesis compara 3 frameworks en diferentes aspectos de desarrollo, estos aspectos son: Comparación y configuración del framework en general, comparación de las estructuras y componentes, comparación de velocidades de aplicación. Los usuarios interactúan a través de la interfaz del navegador, sin ejecutar la aplicación directamente en sus computadoras locales.

Análisis y resultados

Comparación y configuración del framework en general

Como se evidencia en la Tabla V, Angular, es un framework pesado y grande con un gran soporte, se ejecuta en el lenguaje de programación de TypeScript. Este framework se puede clasificar en la categoría de frameworks más pesados con un tamaño de 284 MB para un proyecto por defecto, además, este proyecto tiene una gran variedad de dependencias instaladas en la base del proyecto. La segunda tecnología escogida, React, una librería de Javascript pero también tiene su propio principio de funcionamiento de componentes y manejo de la estructura HTML, esta librería ocupa 188 MB de espacio en memoria. Finalmente, se escogió Blazor, que pertenece a la categoría de framework, ya que tiene muchas dependencias, este

Tabla V. Datos básicos de los frameworks[18].

Característica/Framewok	Angular	Blazor webAssembly	React
Categoría	Framework	Framework	biblioteca Javascript
Año de invención	2016	2020	2013
Lenguaje de programación	TypeScript	C#	JavaScript/ Mecanografiado
Entorno de ejecución	Node.js	.NET Core	Node.js
Producto	En producción	Aún en proceso de desarrollo	En producción

framework implementa el lenguaje de programación C#, el cual se traduce a WebAssembly. En cuanto al tamaño que ocupa Blazor, se puede decir que es relativamente bajo con 1,42 MB de espacio en memoria secundaria.

Comparación de estructuras y componentes. En esta comparación se determinó que los componentes de Angular son los más complejos, extendiendo la funcionalidad de un componente entre múltiples archivos, donde el principio del sistema de componentes se basa en referencias de otros archivos en un único archivo de TypeScript. Los componentes de Angular entran en su ciclo de vida a través de ocho fases. Por parte de React, tiene una estructura simple de componentes, dónde sólo se requiere de código Javascript para el archivo del componente, además, un componente en React funciona devolviendo una estructura HTML como resultado, en este caso, el ciclo de vida de los componentes sólo tiene 3 fases. Finalmente, por parte de los componente de Blazor o componentes razor, se definen en un único archivo que contienen código HTML y C# para definir la lógica de la interfaz de usuario. Los componentes razor constan de 4 etapas en su ciclo de vida, la tabla VI muestra el cuadro comparativo de las características de los componentes en cada framework estudiado.

Comparación de velocidades de la aplicación. Para esta comparación, se midió las velocidades de inicio de 3 proyectos desarrollados en cada framework, específicamente, se midió la velocidad el tiempo que les llevó a todos los módulos en cargarse y el tiempo que le tomó a un compilador en particular en traducir el código y renderizar los elementos en el sitio web. La ejecución de los proyectos se realizaron 3 veces, para tener en cuenta

Tabla VI. Comparación de componentes y ciclos de vida[18].

Propiedad/componente	Angular	Blazor	React
Numero de Archivos	2-5	1	2-1
Renderizado del código original	Referencias	Inyección de código en estructura HTML	Retorno de código HTML a través de funciones Javascript
Número de fases	8	3	4

la mejora de la velocidad debido al caché. Los datos obtenidos fueron los siguientes:

Como se evidencia en la Tabla VII Angular aproximadamente dedica la mitad de tiempo que React y Blazor, esto se debe a que Angular tiene una gran diversidad de dependencias, Además de esto, el hecho de que Angular implemente Typescript como lenguaje de programación, es un factor que afecta la velocidad de carga de la aplicación, debido a que el código TypeScript debe convertirse primero a código Javascript a través del compilador tsc. Por parte de React, el resultado obtenido en el tiempo fue significativamente menor, ya que React es una librería de Javascript, lo cual hace que React sea más rápido que Angular. Finalmente, Blazor demostró que es un framework muy compacto que ocupa mucho espacio en disco, sin embargo, tiene simplicidad para construir componentes, además, Blazor fue el framework en obtener el menor tiempo, un factor que puede afectar a este comportamiento es el uso del estándar webassembly para ejecutar la aplicación en el navegador.

Tabla VII. Comparaciones de velocidad de los frameworks[15].

Framework/experimento	Primera vez	Segunda vez	Tercera vez	Promedio
Angular	4.9	4.9	4.9	4.9
React	2.5	2.3	2.3	2.4
Blazor	2.5	2.4	2.2	2.3

Comparación de la integración de recursos de terceros. Para esta comparación se desarrolló tres aplicaciones con las mismas características de funcionamiento, una en cada framework, donde se evaluó cuantas librerías externas se necesitaron para desarrollar dicha aplicación, la siguiente tabla muestra los resultados obtenidos.

Cómo se puede observar en la tabla VIII, en Angular sólo se usó una dependencia externa, lo que confirma que este framework tiene una gran variedad de funcionalidades que ya vienen incluidas. Con la librería React se usó 17 dependencias externas, lo cual confirma que React está lejos de tener las descripciones de un framework. En cuanto a Blazor, sólo se necesitó 6 dependencias externas, a pesar de tener una cantidad relativamente pequeña, se debe considerar que C# aún no tiene soporte adecuado como el de Javascript para interactuar con el navegador.

Tabla VIII. Comparación del uso de recursos de terceros en aplicaciones por marco[18].

Estructura	Angular	React	Blazor
Numero externo de librerías	1	17	6

Comparación de la dificultad del desarrollo de aplicaciones. Para esta comparación se tuvo en cuenta la bibliografía consultada en esta tesis junto con la experiencia de desarrollo de los autores, este estudio determinó que Angular requiere mucho conocimiento y experiencia, lo cual hace que este framework tenga una alta dificultad de aprendizaje con una alta curva de aprendizaje, además, este framework tiene una gran cantidad de módulos y opciones de personalización[19].

Por otro lado, para aprender React se requiere menos necesidad de grandes conocimientos y experiencia, así mismo, una gran ventaja que tiene esta librería es que si un desarrollador ya domina el lenguaje de programación JavaScript, puede dominar React con relativa rapidez. Esta librería ofrece la flexibilidad de desarrollar aplicaciones teniendo la opción de escoger JavaScript o TypeScript como lenguaje de programación[19].

En el caso de Blazor, se determinó que presenta un desafío para evaluar su dificultad objetiva, ya que esto depende en gran medida del conocimiento del desarrollador sobre el ecosistema ASP.NET y el entorno .NET. Para aquellos familiarizados con C# y expertos en .NET, Blazor resulta más accesible, pero puede representar un obstáculo mayor para desarrolladores sin estas habilidades.

7. Tomado de: Evaluación de Blazor, una comparativa de un framework web.

Esta tesis fue elaborada en la Universidad de Linköping AIF de Suecia en el año 2020[20]. Se realizó una evaluación del framework Blazor, para esto, el autor tomó como referencia artículos sobre la evaluación de frameworks web para utilizar los siguientes criterios de evaluación: documentación, líneas de código, tamaño de la comunidad, el uso del framework, madurez del framework, frescura del framework, soporte del navegador, y costo del framework. Además, teniendo en cuenta que Blazor implementa C# en lugar de JavaScript, se tomó un noveno criterio para comparar Javascript con C#. Para comparar Blazor se tomó como contraparte el framework Angular y Ember.js.

Análisis y resultados

Criterio de Documentación. Para este criterio, el autor determinó que la documentación de Angular es notablemente más grande que la documentación de Blazor, por una parte la documentación de Angular cuenta con 274.171 palabras y Blazor con 94.571 palabras.

Criterio de líneas de código. En este criterio se comparó Blazor con Angular, en este criterio se determinó que Blazor sobresale en este aspecto, esto es debido a que su sintaxis para escribir un componente es compacta y comprensible.

Criterio de tamaño de la comunidad. Para este criterio se determinó que aunque Blazor no alcanza las dimensiones de comunidades como Angular, su base de contribuyentes y la actividad en comunidades como Stack Overflow sugieren un crecimiento y una aceptación notable.

Criterio del uso del framework. Se evaluó este criterio comparando el número de repositorios en GitHub para Blazor, Angular y Ember.js. Blazor mostró 9.000 repositorios, mientras que Angular contaba con 681.000 repositorios, y Ember.js presentaba 33.000 repositorios.

Criterio de madurez del framework. En este criterio se comparó Blazor y Angular, donde se determinó que Blazor es un framework relativamente nuevo en comparación con Angular, es por esto que en este criterio Angular se considera el framework con mayor madurez.

Criterio frescura del framework. En una comparativa entre marcos web, Blazor se posiciona favorablemente. A pesar de la competencia con marcos contemporáneos como Angular o React, Blazor tiene una ventaja sutil gracias a sus sólidas bases tecnológicas, como WebAssembly.

Criterio soporte del navegador. Para este criterio, se determinó que Blazor es compatible con todos los navegadores que admiten los estándares web modernos, y una comparación con otros frameworks web modernos no perjudicará ni afectará a Blazor.

Criterio del costo del framework. En cuanto al costo, Blazor se ofrece de forma gratuita o a un costo razonable para la mayoría de los usuarios.

Los gastos potenciales derivan del proceso de desarrollo e implementación del software, aspectos que suelen ser inherentes a proyectos más amplios, sin importar el marco elegido.

Criterio de comparación entre Javascript y C#. Para esta comparación el autor determinó que a pesar de que Javascript es un lenguaje en el cual un desarrollador puede producir un código rápidamente, este lenguaje es propenso a errores no detectados que son causados por escritura incorrecta.

Teniendo en cuenta, los criterios descritos anteriormente, se determinó cuáles de estos son favorables para el framework Blazor, donde se determina que la favorabilidad significa que Blazor es una opción viable para adoptarlo como framework web, los resultados de la favorabilidad de los frameworks se pueden ver en la Tabla IX.

Tabla IX. Resultado de favorabilidad de Blazor[20].

Criterion	Evaluation
Documentation	Favourable
Lines of Code	Favourable
Community Size	Favourable
Framework Usage	Unfavourable
Framework Maturity	Unfavourable
Framework Freshness	Favourable
Browser Support	Favourable
Framework Cost	Favourable
JavaScript or C#	Ambiguous

Se concluye en esta tesis que Blazor muestra un rendimiento positivo en seis de los ocho criterios evaluados, destacando su eficiencia en el uso de C# y líneas de código. Aunque su documentación difiere de Angular, es extensa, y la comunidad que lo respalda, junto con .NET, es sólida. La juventud de Blazor es su principal inconveniente, pero se ve mitigada por la estabilidad de .NET y el respaldo de una creciente comunidad. Aunque actualmente no hay sitios web a gran escala creados con Blazor, la evaluación sugiere que elegirlo para un nuevo proyecto no es arriesgado, ya que está bien afinado dada su relativa novedad en el panorama de desarrollo de aplicaciones web.

IV. Comparativa de los casos de estudio

Los casos de estudio encontrados abordaron el estudio de los frameworks de diferentes maneras, unos compararon el rendimiento de Blazor vs otros frameworks haciendo uso de diferentes métricas de rendimiento y otros evaluaron aspectos generales, como las líneas de código, popularidad, librerías, comunidad, etc. Por otra parte, el siguiente cuadro comparativo da a conocer las características de los frameworks comparados en las tesis encontradas.

A Partir de la Tabla X, en términos de rendimiento, mantenimiento, escalabilidad, velocidad y facilidad de desarrollo, se puede observar lo siguiente:

Tabla X. Tabla comparativa de Framework[21].

Tecnología/ Framework	Rendimiento	Mantenimiento	Escalabilidad	Velocidad	Facilidad de desarrollo
Blazor	Medio	Bajo	Alto	Medio	Alto
React	Alto	Alto	Muy Alto	Alto	Muy Alto
Svelte	Alto	Medio	Alto	Muy Alto	Muy Alto
Angular	Medio	Medio	Medio	Medio	Medio
Vue	Alto	Alto	Alto	Alto	Alto
Ember,js	Medio	Medio	Medio	Bajo	Bajo

- Sólido y una buena escalabilidad. Su mantenimiento y velocidad son aceptables, y proporciona una facilidad de desarrollo razonable.

- **React:** Destaca en rendimiento, escalabilidad y velocidad. Es relativamente fácil de mantener, y la facilidad de desarrollo es bastante alta.

- **Svelte:** Ofrece un buen rendimiento y velocidad, junto con una escalabilidad y facilidad de desarrollo notables. El mantenimiento es manejable.

- **Angular:** Proporciona un buen rendimiento y escalabilidad, aunque su velocidad puede ser un poco menor en comparación con otras. El mantenimiento y la facilidad de desarrollo son moderados.
- **Vue:** Destaca en facilidad de desarrollo y ofrece un rendimiento y escalabilidad sólidos. La velocidad es razonable, y el mantenimiento es manejable.

- **Ember.js:** Mientras que mantiene un rendimiento y escalabilidad aceptables, puede tener una velocidad y facilidad de desarrollo más moderadas. El mantenimiento es razonable.

En general, la elección entre estas tecnologías/ frameworks dependerá de las prioridades específicas del proyecto. Si se busca un alto rendimiento y escalabilidad, React y Vue son buenas opciones, mientras que Svelte ofrece una combinación sólida de rendimiento y facilidad de desarrollo. Angular y Ember.js son opciones más robustas pero pueden requerir un mayor esfuerzo en mantenimiento y desarrollo.

V. Conclusiones

- Blazor ofrece una ventaja para los desarrolladores al permitirles utilizar el lenguaje familiar de C# para desarrollar aplicaciones tanto en el front-end como en el back-end.

- A pesar de que Blazor WebAssembly obtuvo un desempeño bajo en algunas métricas, en algunos casos de estudio, se mantuvo consistente en los datos tomados, esto sig-

nifica que se mantiene estable en las pruebas realizadas.

- Blazor destaca por posibilitar una escritura más compacta de componentes en comparación con React, ya que permite expresar la lógica de un componente con menos líneas de código.

- Blazor es un framework relativamente nuevo con una comunidad pequeña en comparación con React u otros frameworks populares, sin embargo, es un framework en auge con posibles mejoras que le pueden permitir competir con los frameworks basados en Javascript.

- El cuarto caso de estudio muestra que cada framework tiene un tiempo de ejecución del script de renderizado diferente según la estructura del componente que utiliza. Por lo tanto, es importante tener en cuenta las estrategias de renderizado que cada framework ofrece para compararlos de manera justa y elegir el más adecuado para cada aplicación web.

- Las dos primeras tesis revisadas señalan la escasez de investigaciones que realicen comparaciones exhaustivas entre Blazor y frameworks establecidos como React, Angular y Svelte. Aunque existen algunos estudios comparativos, su número es limitado, lo que dificulta contar con fuentes confiables y amplias para obtener conclusiones definitivas sobre las ventajas y desventajas de Blazor en relación con los mencionados frameworks. Este vacío en la investigación destaca la necesidad de más estudios detallados y rigurosos que permitan una evaluación más completa y precisa de Blazor en comparación con sus contrapartes bien establecidas en el desarrollo de aplicaciones web.

Referencias

[1] "Blazor de ASP.NET Core". Microsoft Learn: Build skills that open doors in your career. Accedido el 9 de enero de 2024.[En línea]. Disponible: https://learn.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-8.0

[2] "Ventajas de crear tu web con Single Page Applications (SPAs) - Making Science". Making Science. Accedido el 9 de enero de 2024.[En línea]. Disponible: https://www.makingscience.es/blog/ventajas-de-crear-tu-web-con-single-page-applications-spas/

[3] Does your web app need a front-end framework? Max Pekarsky Febrero 3, 2020 Accedido el 15 de octubre de 2023.[En línea] Disponible: https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/

[4] J. Engstrom, «Web Development with Blazor: A Hands-on Guide for .NET Developers to Build Interactive UIs with C#,» Reino Unido: Packt Publishing, 2021.

[5] Blazor de ASP .NET Core. Accedido el 15 de Octubre de 2023.[En línea] Disponible: https://learn.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-7.0

[6] "Blazor de ASP.NET Core," Microsoft.com, Accedido el 30 de Noviembre de 2023.[En línea] Disponible: https://learn.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-7.0

[7] C. Sainty, «Blazor in Action,» Simon and Schuster, 2022.

[8] Modelos de Hospedaje en Blazor, Accedido el 12 de Diciembre de 2023.[En línea] Disponible: https://learn.microsoft.com/es-es/aspnet/core/blazor/hosting-models?view=aspnetcore-8.0

[9] Gracia G. Cesar, Infraestructura o plataforma como servicio para despliegues continuos de aplicaciones web en Blazor. 2021.

[10] Lock, A. *ASP. NET core in Action*. Simon and Schuster.2023.

[11] Reiser, M. y Bläser, L, 24 de octubre de 2017. Acelerar JavaScript Aplicaciones mediante compilación cruzada en WebAssembly.VMIL'17: Asociación de Maquinaria de Computación, 10-17.

[12] Haas, A., Rossberg, A., Schuff, DL, Titzer, BL, Holman, M., Gohman, D.,Wagner, L., Zakai, A. y Bastien, J. (junio de 2017). Actualizando la web con WebAssembly.PLDI 2017: Actas de la 38.a Conferencia ACM SIGPLAN sobre diseño e implementación de lenguajes de programación, 185-200.

[13] Wang, S., Ye, G., Li, M., Yuan, L., Tang, Z., Wang, H., Wang, W., Wang, F., Ren, J., Fang, D. y Wang, Z.,2019. Aprovechando WebAssembly para la virtualización de código JavaScript numérico.

[14] S. Nilsson, Evaluation of the Blazor and Angular frameworks performance for web applications, 2021.

[15] O. Köping and E. Persson, Evaluation of the Blazor framework: A comparison between Blazor and React, 2021.

[16] R. Ollila, N. Mäkitalo, and T. Mikkonen, Modern Web Frameworks: A Comparison of Rendering Performance, Journal of Web Engineering, 2022.

[17] K. Kozak and J. Smołka, «Analysis of the Blazor framework in client-hosted mode,» Journal of Computer Sciences Institute, vol. 16, pp. 269–273, 2020.

[18] R. Tkalčič, «Primerjava razvoja aplikacij med blazor, react in angular: diplomsko delo,» Diplomsko delo, Univerza v Mariboru, Maribor, 2022.

[19] "Angular vs ReactJS: Which One to Choose for Frontend Development?" Homepage. Accedido el 11 de enero de 2024.[En línea]. Disponible: https://www.spaceotechnologies.com/blog/angular-vs-react/

[20] SANDBERG, Erik. Evaluating Blazor: A comparative examination of a web framework. 2021.

[21] "Los Frameworks Web más populares según el Informe de Stack Overflow 2023. - Tecnología, ciencia y educación." Azul Web. Accedido el 13 de enero de 2024.[En línea]. Disponible: https://www.azulweb.net/los-frameworks-web-mas-populares-segun-el-informe-de-stack-overflow-2023/