VOL. 5 Nº2 - 2023

IBARRA - ECUADOR

Facultad de Ingeniería en Ciencias Aplicadas
Innovation & Development in Engineering and Applied Science

IDEAS

Integración de componentes Enterprise Java Beans y Front End Angular para analizar el desempeño de acceso a datos

http://doi.org/10.53358/ideas.v5i2.943

Mauricio Rea Peñafiel , Tamia Maldonado Arias, Antonio Quiña Mera

Universidad Técnica del Norte, Ibarra, Ecuador

Fecha de envío, agosto 3/2023 - Fecha de aceptación, septiembre 26/2023 - Fecha de publicación, septiembre 28/2023

Resumen: La plataforma Java EE (Enterprise Edition) permite el desarrollo de software de tipo empresarial, cubriendo todas las capas que una aplicación monolítica requiere, pero carece de funcionalidades reactivas de manera nativa, lo que ocasiona un menor desempeño en la presentación de información al usuario final. Este problema afecta a los usuarios, porque la información que se visualiza en sus pantallas probablemente ya está desactualizada. En esta investigación se plantea comprobar la integración entre Java EE (plataforma backend) y Angular (framework frontend), con el propósito de mejorar el tiempo de respuesta para el usuario. Se utiliza el método Design Science Research como ruta principal de la investigación para determinar el problema, los antecedentes, la situación actual y los objetivos del proyecto propuesto. La parte teórica incluye información sobre plataformas tecnológicas y la metodología para el desarrollo del software propuesto. Se implementa una capa de servicios REST para proporcionar información de un sistema utilizado como caso de estudio en la Universidad Técnica del Norte, que se denomina SIAD (Sistema Integrado de Actividades Docentes), dicho servicio es consumido por la aplicación web Angular. Por otro lado, se ejecutan las pantallas JSF, las cuales consumen la información proporcionada por los componentes EJB. Finalmente, se ejecutan los dos tipos de clientes web (JSF y Angular) y se valida el desempeño utilizando el estándar ISO/IEC 25010. Se puede concluir de manera global que, los componentes reactivos de Angular mejoran el indicador de desempeño del sistema informático en un 21%.

Palabras clave: EJB, API REST, ISO 25000, JSF, Angular.

Abstract: The Java EE (Enterprise Edition) platform enables the develop-ment of enterprise software, covering all the layers required by a monolithic application. However, it lacks native reactive functionalities, resulting in reduced performance in delivering information to the end user. This issue impacts users because the information displayed on their screens is likely outdated. This research aims to verify the integration between Java EE (backend platform) and Angular (frontend framework) with the purpose of improving response time for the user. The Design Science Research method is employed as the primary research approach to identify the problem, back-ground, current situation, and objectives of the proposed project. The theo-retical portion includes information about technological platforms and the methodology for the development of the proposed software. A REST ser-vice layer is implemented to provide information from a system used as a case study at the Technical University of the North, referred to as SIAD (In-tegrated System of Teaching Activities). This service is consumed by the Angular web application. Additionally, JSF screens are executed, which consume information provided by EJB components. Finally, both types of web clients (JSF and Angular) are executed, and performance is validated using the ISO/IEC 25010 standard. It can be globally concluded that Angu-lar's reactive components enhance the performance indicator of the comput-er system by 21%.

Keywords: EJB, API REST, ISO 25000, JSF, Angular.

Autor de correspondencia:
Mauricio Rea , mrea@utn.edu.ec

Introducción

Las aplicaciones empresariales manejan grandes cantidades de datos y requieren integración de sistemas distribuidos. Los desarrolladores actuales reconocen la importancia de tener aplicaciones portables, transaccionales y distribuidas, que aprovechen la velocidad, seguridad y confiabilidad del lado del servidor. Estas aplicaciones contienen la lógica empresarial, se gestionan centralmente y se integran con otros sistemas. En la industria de tecnología de la información, se busca diseñar, construir y producir aplicaciones empresariales de forma más eficiente, rápida y con menos recursos [1]. Es importante que reaccionen de manera inmediata a las solicitudes de los usuarios, si bien las aplicaciones de Java Enterprise Edition tienen una arquitectura monolítica consistente, modular y madura [2], no activan automáticamente los métodos de sus componentes ante las actualizaciones de información por parte de los usuarios.

Los métodos de los componentes de JavaEE tienen una baja reactividad ante las modificaciones de datos del usuario, debido a su enfoque clásico, lo que resulta en demoras significativas entre solicitudes. La programación reactiva ofrece ventajas al procesar secuencias, eventos o mensajes [3]. Además, el alto número de usuarios concurrentes y la alta actualización de datos consumen recursos y pueden dejar la vista o el modelo de datos desactualizados. Adicionalmente, los fallos no son manejados de manera apropiada, lo que puede afectar la visualización de la información buscada.

La evolución tecnológica ha generado múltiples herramientas y arquitecturas para el diseño y desarrollo de sistemas y aplicaciones web con mejores resultados y que los modelos monolíticos puedan mejorarse, superando sus limitaciones de escalabilidad y rendimiento [4]. Las arquitecturas de servicios web fomentan la colaboración entre sistemas.

La tecnología API REST requiere de los servicios de un backend [5], facilitando los cambios en cada uno sin afectar su funcionalidad principal. La API REST permite crear nuevas funcionalidades, tanto en la vista de la aplicación como en la capa de servicios, sin afectar su estructura. Además, mejora la experiencia del cliente al recibir información plana y reduce el tiempo de transmisión y el consumo de recursos físicos.

La alta concurrencia de usuarios y por consiguiente la actualización de información, conduce a un alto uso de recursos y la falta de un flujo constante de datos, lo que puede mantener la vista o el modelo de datos desactualizados en ciertos momentos y como consecuencia, los usuarios no podrán visualizar la información buscada.

Es así como, en este trabajo, se toma como caso de estudio un software de planificación académica, mismo que está implementado en la Universidad Técnica del Norte y que ha sido utilizado con fines académicos para el desarrollo de varios proyectos. El software mencionado se denomina SIAD (Sistema Integrado de Actividades Docentes) y está construido sobre una plataforma Java Enterprise, con arquitectura monolítica y con las tecnologías JPA, EJB y JSF. Se plantea realizar la integración entre el backend (JavaEE con componentes EJBs) y a través de un API REST pueda brindar información a una aplicación de tipo frontend desarrollada en la plataforma Angular, y poder determinar una posible optimización y mayor eficiencia. Angular, utilizando la librería RxJS que es una implementación destacada de la programación reactiva en JavaScript, ofrece varias funcionalidades que permiten que el software sea responsivo, elástico y orientado a mensajes [6].

Esto se podrá corroborar mediante la aplicación de la norma ISO/IEC 25010 mediante la subcaracterística de eficiencia de desempeño [7]. Esta norma es parte de un conjunto de normativas que permiten evaluar la calidad del software[8].

Materiales y métodos

Método Design Science Research

Se ha utilizado el método DSR (Design Science Research), el cual brinda los lineamientos para determinar y desarrollar las actividades necesarias para cumplir con el objetivo de investigación [9].

Tabla 1. Diseño de la investigación.

Actividad	Componentes	Sección del artículo
Diagnóstico del problema	Problema, Objetivo.	Introducción, Diseño de la investigación
Fundamento teórico	Arquitectura del software, metodología de desarrollo, ISO/IEC 25000.	Diseño de la investigación.
Diseño de la propuesta: Integración entre aplicación frontend y backend.	Definición de requerimientos, diseño de la arquitectura, implementación.	Diseño de la investigación.
Evaluación	Evaluación del desempeño de los componentes EJBs mediante un API REST.	Resultados.

Fundamentación teórica

Java Platform, Enterprise Edition (Java EE) es la plataforma para desarrollar software empresarial y que es utilizada por una gran comunidad [10]. Se define con aportaciones de expertos, grupos de usuarios de Java, la comunidad de código abierto y organizaciones comerciales. Cada versión presenta nuevas características que mejoran la portabilidad de las aplicaciones y la productividad del desarrollador [11].

JSF es un framework de desarrollo web Java que se centra en los componentes del lado del servidor. Ofrece una manera estandarizada y personalizada de crear interfaces de usuario en aplicaciones web empresariales, reduciendo el tiempo de desarrollo y mantenimiento. Este tipo de aplicaciones se denominan Page-Flow Web Application [12], puesto que son un conjunto de páginas que van interactuando, dependiendo del flujo de navegación proporcionado por las acciones de un controlador frente a las interacciones con los usuarios. Otorga soluciones estándar para problemas comunes como validación, navegación, plantillas y flujos de páginas, todo esto debido a que implementa el patrón de diseño MVC (Model View Controller) [13]. Si bien, este framework es el estándar de facto para desarrollo Java Web, el objetivo de este trabajo es lograr prescindir de la vista y controlador JSF y determinar el desempeño de la capa modelo cuando se integre mediante API REST hacia un frontend. El ciclo de vida de JSF consta de diez fases que procesan una solicitud, desde su recepción en el servidor hasta la generación de una página HTML [14]. Cada fase tiene un objetivo específico dentro del proceso. Es por lo que, se busca una alternativa a este

tipo de procesamiento monolítico para verificar el desempeño de los componentes de modelo.

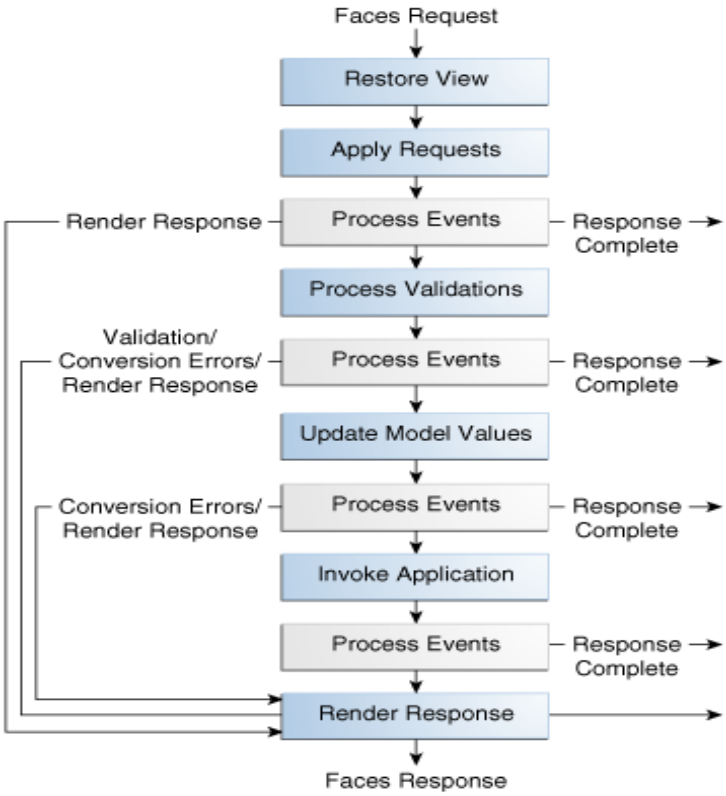

Fig. 1. Ciclo de vida estándar de componentes JSF [14]

La capa model (modelo) sería el conjunto de componentes que harán las funciones de backend, que se encargan de implementar las funciones internas del servidor como tomar datos, procesarlos y enviar la respuesta al usuario [15].

Angular es una plataforma para crear aplicaciones web combinando plantillas declarativas, inyección de dependencia y herramientas integradas. Permite a los desarrolladores crear aplicaciones para web, dispositivos móviles y escritorio. El código de Angular se puede escribir en JavaScript moderno, utilizando inyección de dependencia y decoradores para metadatos [16]. El ciclo de vida de sus componentes es más reducido que el de JSF [17].

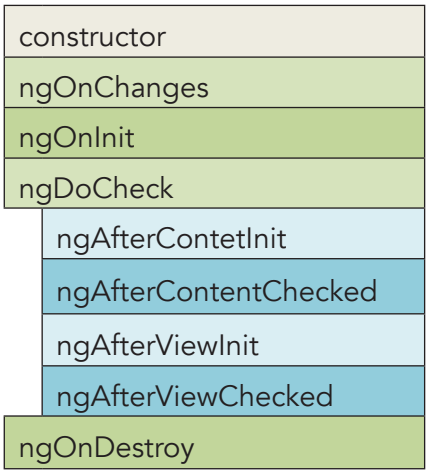

Fig. 2. Ciclo de vida de Angular [17]

La librearía de Angular que permite implementar funciones asíncronas y basadas en eventos es ReactiveX, utilizando secuencias observables [18]. Extiende el patrón observador y proporciona operadores para componer secuencias de manera declarativa. Además, se encarga de manejar consideraciones como subprocesos, sincronización y estructuras de datos concurrentes [19].

La eficiencia de Java EE como de Angular en la ejecución de las pantallas de consulta se evalúa mediante la característica de desempeño de la norma ISO 25010, que abarca el comportamiento temporal, la utilización de recursos y la capacidad. El comportamiento temporal se refiere a los tiempos de respuesta y procesamiento, el uso de recursos se relaciona con las cantidades y tipos de recursos utilizados, y la capacidad se refiere al cumplimiento de los límites máximos de un parámetro [7].

Las métricas de calidad para la característica de eficiencia en el desempeño son [20]:

Tabla 2. Métricas de calidad – Característica eficiencia en el desempeño [20]

Subcaracterística	Métrica	Propósito	Aplicación	Fórmula / Variables
Comportamiento del tiempo	Tiempo de respuesta	Determinar el tiempo para completar una tarea	Tiempo desde que se envía la petición hasta obtener la respuesta	X = B - A A= Tiempo de envío de petición B = Tiempo en captar la primera respuesta
	Tiempo de espera	Determinar el tiempo desde el envío de una petición para que inicie un trabajo, hasta que lo completa	Tiempo cuando se inicia un trabajo y el tiempo en completar el trabajo	X = B - A A= Tiempo de inicio de una actividad B = Tiempo en completar la actividad
	Rendimiento	Número de tareas procesadas por unidad de tiempo	Contar el nro de tareas completadas en un lapso	X = A/T A= Nro de tareas completadas T = Intervalo de tiempo Condición: T > 0
Uso de recursos	Uso de CPU	Determinar el tiempo de CPU usado para completar una tarea	Tomar el tiempo de operación y la cantidad de tiempo de CPU que se usa para realizar una tarea	X = B-A A= La cant. de tiempo de CPU que es usado para realizar una tarea B = Tiempo de operación Condición: B > 0
	Utilización de la memoria	Determinar la memoria usada para completar una tarea	Cant. total de memoria que es usado para realizar una tarea	X = B-A A = Cant. de memoria usada para completar una actividad B = Cant. total de memoria Condición: B > 0

Marco de trabajo SCRUM

Scrum, una metodología ágil, ha ganado popularidad por su enfoque en aumentar la velocidad de desarrollo, fomentar una cultura centrada en el rendimiento y crear valor. Facilita una buena comunicación [21] de los resultados en todos los niveles, mejorando el desarrollo individual y la calidad del tiempo de vida mediante la creación de ciclos breves, llamados "Sprints" [22].

Diseño e implementación de la propuesta

La arquitectura propuesta define una nueva capa de componentes que encapsulan la funcionalidad de la capa modelo, que está conformada por EJBs (Enterprise Java Beans) [23] mediante un API REST. Para que exista la correcta serialización / deserialización de la información, se implementó el patrón de diseño DTO (Data Transfer Object), lo que permite que la información obtenga el formato adecuado para ser transferido mediante HTTP y las operaciones fundamentales REST: POST, PUT, DELETE y GET.

Los webservices REST serán consumidos por la aplicación frontend desarrollada con el framework Angular y, para monitorear el comportamiento en el lado del servidor, se obtendrán datos estadísticos a través de la configuración de la librería Java Melody [24].

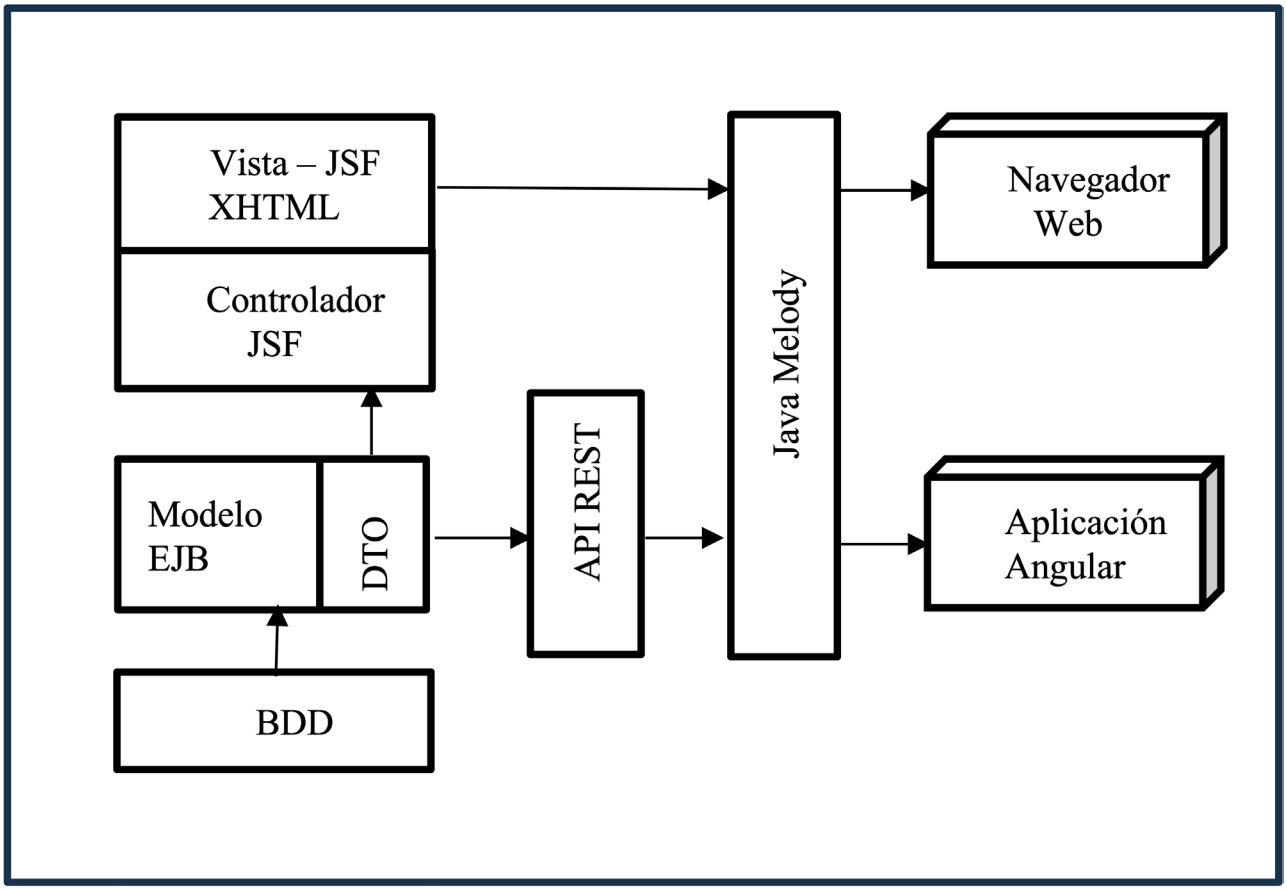

Fig. 3. Arquitectura de la solución propuesta

Resultados

La información obtenida hace referencia a un conjunto de consultas ejecutadas desde el frontend hacia el API REST (operación GET). En general, el consumo de memoria en la ejecución de ambos frontends fue de 230 Mb a 271 Mb, con un promedio de 248Mb y en el uso de procesador, un resultado de entre 595 ms a 2437 ms con un promedio de 942 ms.

Tiempos de respuesta

Para esta métrica, se determina el tiempo para completar una actividad, desde que se envía la petición hasta tener la respuesta:

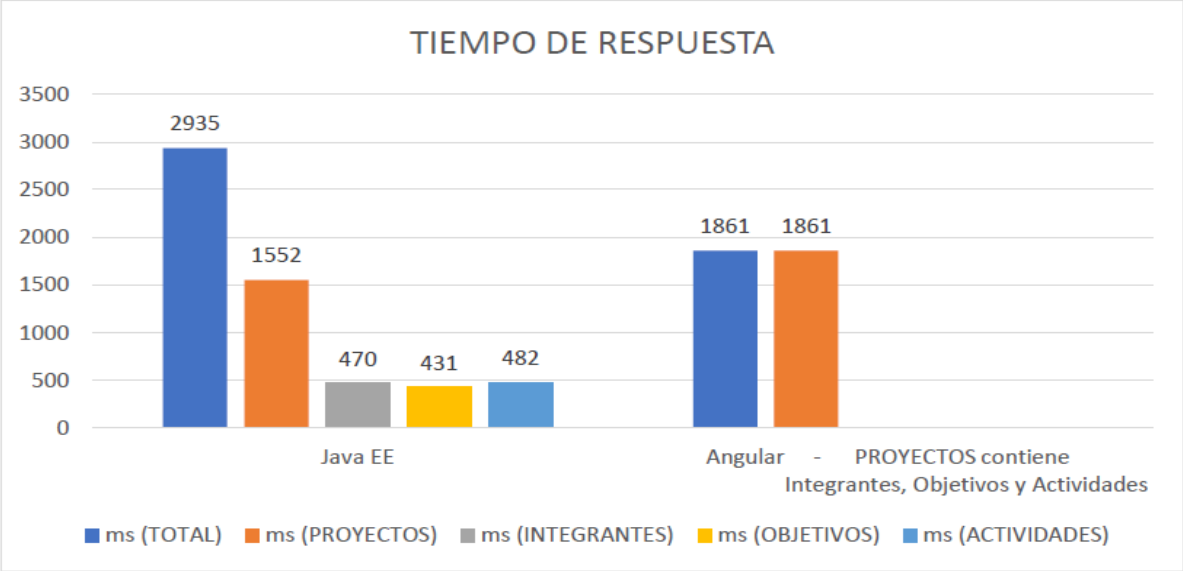

Fig. 4. Comparación tiempo de respuesta cliente JSF vs cliente Angular

Se puede apreciar que, la consulta con componentes clásicos de JSF hacia los métodos de los EJBs toma más tiempo en la sumatoria total. El proceso de transferencia de información es secuencial (no pueden generarse hilos concurrentes de consulta) y se transmiten datos de las entidades: proyectos, integrantes, objetivos y actividades. En cambio, al realizar las mismas consultas desde el frontal Angular, y aprovechando la definición de los componentes DTO, se hace una sola transferencia con toda la información requerida, lo que es óptimo y se ejecuta en un tiempo menor. Esto podría darnos una idea de nuevos paradigmas como GraphQL[25], que determinan un comportamiento parecido en el intercambio consolidado de información.

Rendimiento

En cuanto a la cantidad de unidades de información que procesa el sistema en un período de tiempo se tiene el siguiente resultado:

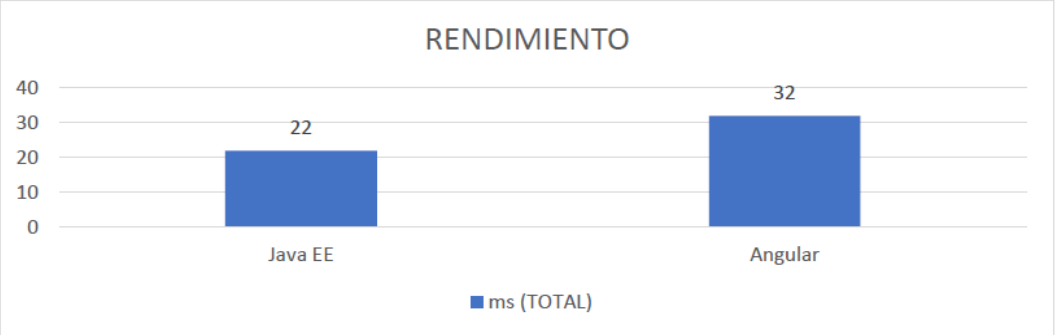

Fig. 5. Comparación de rendimiento

Considerando un mismo período de tiempo, el número de tareas ejecutadas con el frontend Angular es mayor que las completadas en el sistema web JSF.

Utilización de CPU

En esta métrica se determina la cantidad de milisegundos (ms) para completar una misma tarea. Se tiene el siguiente resultado:

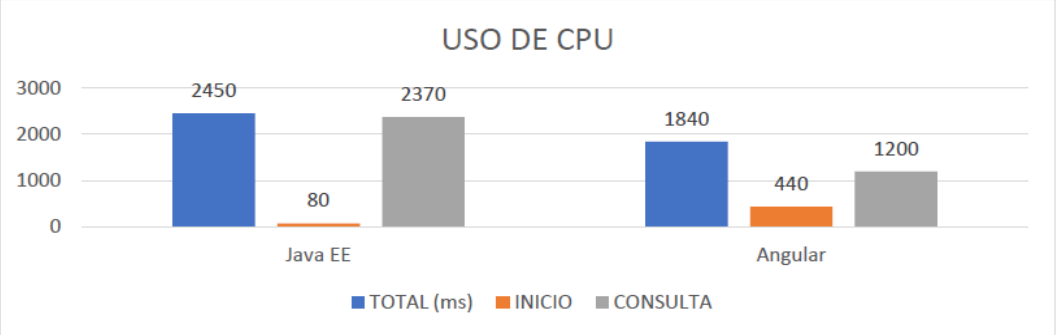

Fig. 6. Comparación en el uso de procesador

El tiempo requerido para completar una solicitud desde el frontal Angular resulta ser más rápido que una misma solicitud desde JSF.

Uso de memoria

Esto se cuantifica en megabytes (Mb) y es el espacio de almacenamiento RAM requerido para realizar una tarea:

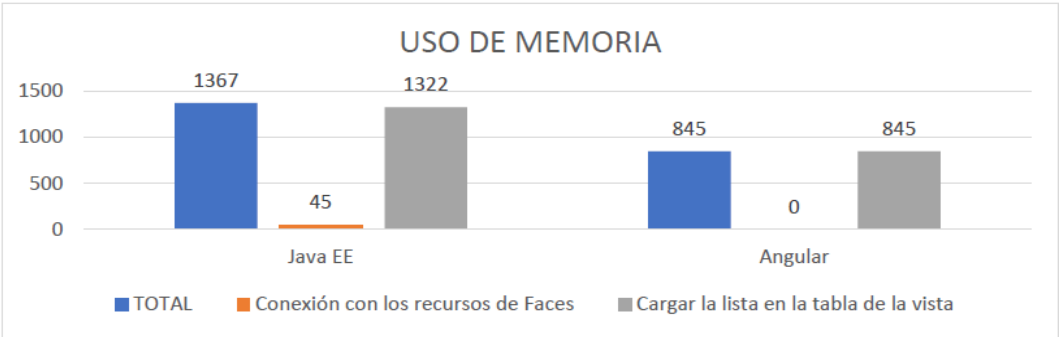

Fig. 7. Comparación del uso de memoria.

Angular utiliza menos memoria para la presentación de información en comparación con JavaEE, puesto que este último necesita conectarse a recursos adicionales para la renderización de contenidos con los componentes Java Server Faces.

Capacidad

Se determina la cantidad de peticiones en línea (online).

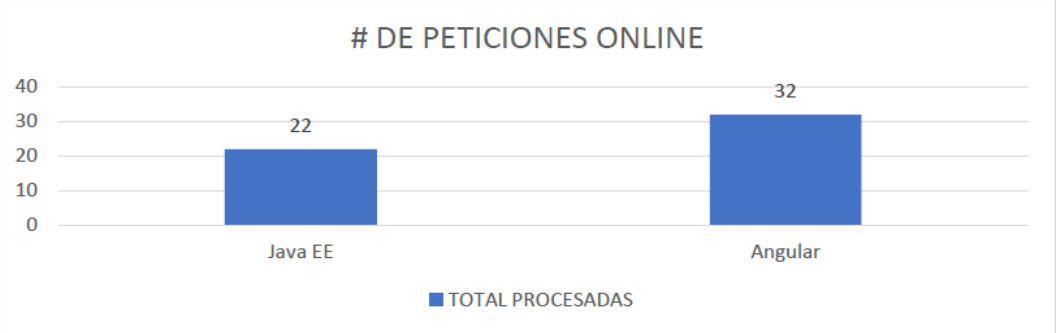

Fig. 8. Comparación de número de peticiones en línea

En la misma unidad de tiempo, el cliente Angular permite lanzar un mayor número de peticiones online. Esto se determina mediante el análisis del comportamiento en el lado del servidor al responder a dichas peticiones.

3.6 Validación de resultados

Se evaluaron las funciones de consulta y presentación de datos de las estructuras correspondientes a "proyectos", utilizando el estándar ISO/IEC 25023:2016, que proporciona medidas de calidad del software en una escala de 1 a 10. Para demostrar el grado de satisfacción, se consideraron los niveles de puntuación propuestos por la Norma ISO/IEC 25040.

Tabla 3. Escala de medición ISO 25040

Escala de medición	Niveles de puntuación	Grado de satisfacción
8,00 – 10,00	Cumple con los requisitos	Muy satisfactorio
5,00 – 7,95	Aceptable	Satisfactorio
1,00 – 4,95	Inaceptable	Insatisfactorio

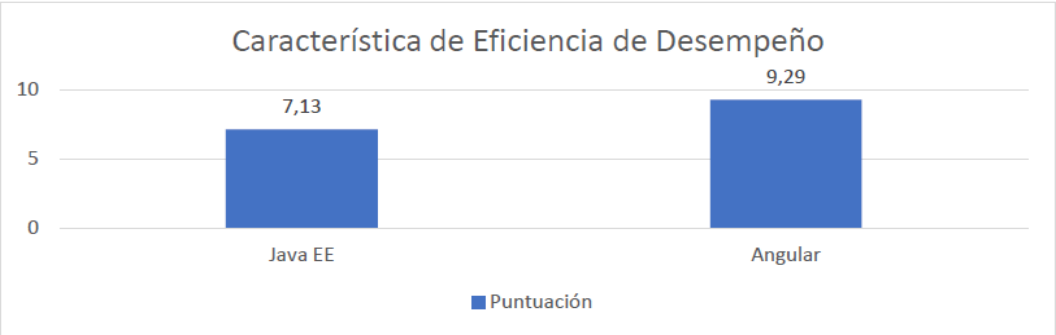

Fig. 9. Comparativa de puntuación JavaEE vs Angular

El detalle de la puntuación obtenida tanto por los componentes Java EE (JSF) como por los componentes Angular se presenta a continuación:

Tabla 4. Eficiencia en el desempeño: caso Java EE

Sub característica	Métrica	Variables		Valor Obtenido	Valor métrica / 10	Final subcaracterística	Total característica
		A	B	X			
Comportamiento del tiempo	Tiempo de respuesta	0	2.9350	2.94	7.07	7.16	7.13
	Tiempo de espera	0.0030	2.9250	2.92	7.08		
	Rendimiento	22		7.33	7.33		
Uso de recursos	Uso de CPU	0.0181	2.4500	2.43	7.57	6.90	
	Uso de memoria	45.00	1322.00	1277.00	6.23		
Capacidad	Número de peticiones online	22		7.33	7.33	7.33	

Tabla 5. Eficiencia en el desempeño: caso Angular

Sub característica Métrica Variables Valor Obtenido Valor métrica / 10 Final subcaracterística Total característica

Sub característica	Métrica	Variables		Valor Obtenido	Valor métrica / 10	Final subcaracterística	Total característica
		A	B	X			
Comportamiento del tiempo	Tiempo de respuesta	0	1.8610	1.86	8.14	8.77	9.29
	Tiempo de espera	0.0030	1.8270	1.82	8.18		
	Rendimiento	32		10.67	10		
Uso de recursos	Uso de CPU	0.0181	1.8500	1.83	8.17	9.08	
	Uso de memoria	0.00	845.00	845.00	10.00		
Capacidad	Número de peticiones online	32		10.67	10.00	10.00	

Discusión

Si bien el stack JPA-EJB-JSF presenta madurez, portabilidad, alta disponibilidad y una simplificación en el desarrollo [26], fue necesario investigar nuevas estrategias que mejoren el rendimiento de la combinación de tecnologías mencionadas, aunque eso implique algún cambio en la arquitectura de componentes. La alternativa JPA-Spring-JSF implica que no se usen componentes EJB [27], por ello, no se la ha considerado para esta investigación.

Ambos frameworks, JSF y Angular, responden a la estrategia de manejo de objetos mediante peticiones-respuestas y pueden integrarse a la capa de los servicios que prestan los componentes de backend denominados Enterprise Java Beans. Otros trabajos hacen el análisis frente a un segundo backend como puede ser Laravel, Spring [28], Python, Nodejs [29] o .NET [30], pero, existen restricciones empresariales donde no se puede cambiar el backend, sino que se debe reutilizar, en este caso se representa por los EJBs.

Un framework java que no se ha considerado en este estudio es JavaFX. Si bien este conjunto de componentes soporta la integración con API REST, sus aplicaciones se direccionan hacia una arquitectura de aplicaciones de escritorio [31]. El interés principal en este estudio son las plataformas web, tanto en la integración como en los frontends.

Conclusiones

Se pudo evidenciar que el framework Angular determina un mejor desempeño en el acceso a la capa de servicios que encapsula el funcionamiento de los EJBs. Esto básicamente queda definido por el tipo de aplicación SPA (Single Page Application) que genera Angular y que se procesa en un entorno de ejecución separado del backend. Para lograr un mejor rendimiento en las aplicaciones de tipo Page-Flow Web Application, se debe considerar cambiar alguno de los siguientes aspectos (específicamente en el uso de JSF): utilizar Beans de tipo @RequestScoped o @ApplicationScoped, lo cual mejorará el rendimiento en tiempo y uso de memoria, pero, implica un posible rediseño en la navegación de las páginas. No se recomienda @ViewScoped porque esta anotación quedará en desuso en la nueva especificación Jakarta EE.

Otra mejora que puede establecerse es un mayor uso del componente <f:ajax> de JSF, de manera que exista una comunicación más granular y exista una menor transferencia de bloques de datos entre el cliente web y el servidor [32].

Si bien podría pensarse en cambiar todo el framework JSF y migrar hacia JSP (Java Server Page), tendría que hacerse un análisis minucioso de las ventajas / desventajas que pueden suscitarse, puesto que JSP determina un mayor tiempo en el diseño e implementación de las páginas web.

Un punto adicional a considerar es que, las páginas web desarrolladas con JSF suelen ser más atractivas al usuario, debido a que usualmente se adiciona la capa de componentes ricos de Primefaces, que le da más usabilidad y mejora la experiencia del usuario, pero significa una carga adicional de procesamiento, lo que disminuye el rendimiento.

Referencias

1. Oracle, The Jakarta® EE Tutorial. Accedido: 1 de agosto de 2023. [En línea]. Disponible en: https://eclipse-ee4j.github.io/jakartaee-tutorial/

2. A. C. Nieto Lemus, Arquitectura por componentes jee, un caso práctico, Gerencia Tecnológica Informática, vol. 14, n.o 38, p. 1, 2015.

3. F. Myter, C. Scholliers, y W. De Meuter, Distributed reactive programming for reactive distributed systems, arXiv preprint arXiv:1902.00524, 2019.

4. A. Quiña-Mera, D. Flores Landeta, X. M. Rea-Peñafiel, y C. Guevara-Vega, Quality Evaluation of a Spring Cloud Microservices Architecture Implementation, en Communication, Smart Technologies and Innovation for Society: Proceedings of CITIS 2021, Springer, 2022, pp. 745-754.

5. B. Văduva y H. Vălean, Designing a Low-Code CRUD framework, Carpathian Journal of Electronic and Computer Engineering, vol. 14, n.o 1, pp. 11-19, ago. 2021, doi: 10.2478/cjece-2021-0003.

6. J. Bonér, D. Farley, Kuhn, y M. Thompson, El Manifiesto de Sistemas Reactivos. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://www.reactivemanifesto.org/es

7. ISO 25010, ISO 25010. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://iso25000.com/index.php/normas-iso-25000/iso-25010

8. E. A. Balseca Chisaguano, Evaluación de calidad de productos de software en empresas de desarrollo de software aplicando la norma ISO/IEC 25000, bachelorThesis, Quito, 2015., 2014. Accedido: 21 de septiembre de 2023. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/9113

9. O. Díaz, J. P. Contell, y J. R. Venable, Strategic Reading in Design Science: Let Root-Cause Analysis Guide Your Readings, en Designing the Digital Transfor-mation, A. Maedche, J. vom Brocke, y A. Hevner, Eds., en Lecture Notes in Com-puter Science. Cham: Springer International Publishing, 2017, pp. 231-246. doi: 10.1007/978-3-319-59144-5_14.

10. N. E. Pilco Guachi, Desarrollo de una aplicación web para el control académico de la Escuela de Educación Básica "Capullitos" aplicando la norma ISO/IEC 9126-2 para determinar el nivel de usabilidad., bachelorThesis, Escuela Superior Politécnica de Chimborazo, 2018. Accedido: 21 de septiembre de 2023. [En línea]. Disponible en: http://dspace.espoch.edu.ec/handle/123456789/9274

11. L. Saeed, What Is Java EE?, en Introducing Jakarta EE CDI: Contexts and Dependency Injection for Enterprise Java Development, L. Saeed, Ed., Berkeley, CA: Apress, 2020, pp. 1-2. doi: 10.1007/978-1-4842-5642-8_1.

12. P. Späth, Building Page-Flow Web Applications with JSF, en Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional, P. Späth, Ed., Berkeley, CA: Apress, 2019, pp. 53-132. doi: 10.1007/978-1-4842-5079-2_4.

13. J. Juneau, The Basics of JavaServer Faces, en Java EE 8 Recipes: A Problem-Solution Approach, J. Juneau, Ed., Berkeley, CA: Apress, 2018, pp. 103-189. doi: 10.1007/978-1-4842-3594-2_3.

14. Oracle Inc., The Lifecycle of a JavaServer Faces Application. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://javaee.github.io/tutorial/jsf-intro007.html#BNAQQ

15. F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, y T. Toulkeridis, From Monolithic Systems to Microservices: A Comparative Study of Performance, Applied Sciences, vol. 10, n.o 17, Art. n.o 17, ene. 2020, doi: 10.3390/app10175797.

16. Google Inc., Angular - What is Angular? Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://angular.io/guide/what-is-angular

17. Google Inc., Angular lifecycle hook. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://v2.angular.io/docs/ts/latest/guide/lifecycle-hooks.html

18. ReactiveXb, ReactiveX - Observable. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://reactivex.io/documentation/observable.html

19. ReactiveX, ReactiveX - Intro. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://reactivex.io/intro.html

20. T. N. Vaca Sierra, Modelo de calidad de software aplicado al módulo de talento humano del sistema informático integrado universitario – UTN, masterThesis, 2017. Accedido: 20 de septiembre de 2023. [En línea]. Disponible en: http://repositorio.utn.edu.ec/handle/123456789/7457

21. B. Arias y O. Alvear, Análisis del resultado de la implementación de SCRUM, LEAN Y BSC en el proceso de desarrollo de software en la industria del Retail, Revista Perspectivas, vol. 4, n.o 1, Art. n.o 1, feb. 2022, doi: 10.47187/perspectivas.4.1.116.

22. K. S. & J. Sutherland, La Guía Definitiva de Scrum: Las Reglas del Juego, nov. 2020, Accedido: 19 de septiembre de 2023. [En línea]. Disponible en: https://repositorio.uvm.edu.ve/handle/123456789/59

23. Oracle Inc.b, Java EE APIs. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://javaee.github.io/tutorial/overview008.html#BNACL

24. E. Vernat, Home · javamelody/javamelody Wiki · GitHub. Accedido: 2 de agosto de 2023. [En línea]. Disponible en: https://github.com/javamelody/javamelody/wiki

25. A. Quiña-Mera, C. Guevara-Vega, J. Caiza, J. Mise, y P. Landeta, REST, GraphQL, and GraphQL Wrapper APIs Evaluation. A Computational Laboratory Experiment, en Proceedings of International Conference on Information Technology and Applications, S. Anwar, A. Ullah, Á. Rocha, y M. J. Sousa, Eds., en Lecture Notes in Networks and Systems. Singapore: Springer Nature, 2023, pp. 397-407. doi: 10.1007/978-981-19-9331-2_34.

26. F. Pech-May, M. A. Gomez-Rodriguez, y S. U. Lara-Jeronimo, Desarrollo de Aplicaciones web con JPA, EJB, JSF y PrimeFaces, 2012.

27. C. Heredia, Integración de Spring, Hibernate y JSF en el desarrollo de aplicaciones web. Accedido: 3 de agosto de 2023. [En línea]. Disponible en: https://oa.upm.es/38692/

28. S. M. Mafla Flores, Comparativa de los frameworks angular y primefaces para el desarrollo del aplicativo control de materia prima en la empresa Mastercubox S.A., utilizando la metodología Scrum, bachelorThesis, 2019. Accedido: 3 de agosto de 2023. [En línea]. Disponible en: http://repositorio.utn.edu.ec/handle/123456789/9017

29. M. A. Khan, S. Gairola, B. Jha, y P. Praveen, Smart Computing: Proceedings of the 1st International Conference on Smart Machine Intelligence and Real-Time Com-puting (SmartCom 2020), 26-27 June 2020, Pauri, Garhwal, Uttarakhand, India. CRC Press, 2021.

30. J. Cincovic, S. Delcev, y D. Draskovic, Architecture of web applications based on Angular Framework: A Case Study, 2019.

31. J. Ortega, Extensión de una aplicación web destinada al for- talecimiento de clubes escolares matemáticos integrando JavaFX2 y JavaEE6 con servicios web basados en REST, 2013. Accedido: 3 de agosto de 2023. [En línea]. Disponible en: https://core.ac.uk/download/pdf/71419433.pdf

32. P. Späth, Building Single-Page Web Applications with REST and JSON, en Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional, P. Späth, Ed., Berkeley, CA: Apress, 2019, pp. 133-163. doi: 10.1007/978-1-4842-5079-2_5.